Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Gašper Slapničar

Priporočanje nastanitev z uporabo ponudnika strojnega učenja v oblaku

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Zoran Bosnić

Ljubljana, 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Kandidat naj v svojem diplomskem delu razišče dosedanje implementacije in možnosti za implementacijo priporočilnega sistema, ki uporablja algoritme za strojno učenje izbranega spletnega ponudnika. Izbere naj ustrezno platformo in algoritme za priporočanje, formalizira naj problem priporočanja z definiranjem vrednosti matrike preferenc in naj evalvira delovanje razvitega priporočilnega sistema. Uspešnost priporočilnega sistema naj ovrednoti v kontekstu sorodnih rešitev.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Gašper Slapničar, z vpisno številko **63110249**, sem avtor diplomskega dela z naslovom:

Priporočanje nastanitev z uporabo ponudnika strojnega učenja v oblaku

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Zorana Bosnića,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 7. septembra 2015 Podpis avtorja:

Zahvaljujem se svojemu mentorju izr. prof. dr. Zoranu Bosniću za vse konstruktivne predloge, čas, trud, in potrpljenje tekom nastanka tega diplomskega dela. Posebna zahvala tudi somentorju na Institutu "Jožef Stefan" dr. Boštjanu Kaluži, ki me je seznanil z arhitekturo porazdeljenih sistemov in strojnim učenjem ter me znal v pravih trenutkih motivirati in mi obenem pustiti čas za samostojno delo. Hvala prof. dr. Matjažu Gamsu, ki mi je ponudil priložnost za raziskovalno delo na Odseku za inteligentne sisteme. Zahvalil bi se tudi dr. Mitji Luštreku, ki je aktivno sodeloval na projektu in s svojim vpogledom in predlogi prispeval h konstantnemu napredku projekta. Prof. dr. Gamsu se zahvaljujem tudi za dovoljenje za uporabo podatkov tega projekta v diplomskem delu.

Hvala kolegu Leonu Noetu Jovanu, ki si je pogosto vzel čas in mi pomagal bolje razumeti podrobnosti algoritmov matrične faktorizacije.

Na koncu se za vso podporo tekom študija zahvaljujem še vsem svojim bližnjim.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
REST	Representational State Transfer	/
CF	Collaborative Filtering	Skupinsko filtriranje
GPS	Global Positioning System	Globalni pozicijski sistem
IT	Information Technology	Informacijska tehnologija
BLOB	Binary large object	Velik binarni objekt
HDFS	Hadoop Distributed File System	Porazdeljen datotečni sistem Hadoop
YARN	Yet Another Resource Negotiator	Še en pogajalec resursov
RDBMS	Relational Database Management Systems	Sistemi za upravljanje relacijskih podatkovnih baz
RAM	Random Access Memory	Pomnilnik z naključnim dostopom
HDD	Hard Disk Drive	Trdi disk
API	Application Programming Interface	Programski vmesnik
RDD	Resilient Distributed Dataset	Odporna porazdeljena množica podatkov

kratica	**angleško**	**slovensko**
JSON	Javascript Object Notation	Objektna notacija Javascript
NRT	Near Real Time	Skoraj v realnem času
MLaaS	Machine Learning as a Service	Strojno učenje kot storitev
IDE	Integrated Development Environment	Vgrajeno razvijalsko okolje
ML	Machine Learning	Strojno učenje
MVC	Model View Controller	Model pogled krmilnik
MB	Megabyte	Megabajt
AWS	Amazon Web Services	Amazonove spletne storitve
HTTP	Hypertext transfer protocol	Protokol za prenos hiperteksta
SVD	Singular Value Decomposition	Singularni razcep

Povzetek

Priporočilni sistemi so vseprisotna tehnologija na spletu in lahko ključno vplivajo na poslovne rezultate podjetij. V diplomskem delu se soočimo z razvojem produkcijskega priporočilnega sistema za spletno stran, ki ponuja ekološke nastanitve, ki dosegajo določene ekološke standarde (npr. uporaba sončne energije, filtriranje in ponovna uporaba vode, recikliranje odpadkov itd.). Najprej pregledamo tehnologije velikih podatkovnih množic (angl. *big data*) in ponudnike strojnega učenja v oblaku. Nato izberemo najustreznejšo platformo in jo uporabimo za zbiranje podatkov in razvoj priporočilnega sistema, ki vrača priporočila za uporabnika z uporabo algoritma matrične faktorizacije (angl. *Alternating Least Squares, ALS*) ter obenem vrača tudi podobne priporočilne objekte z uporabo Jaccardove podobnosti in evklidske razdalje. Na koncu sistem interno ocenimo na že zbranih podatkih z uporabo statistične mere *Precision@k*. Rezultati evalvacije so pokazali 19% točnost napovedi, kar je bistveno boljše od naključnega priporočanja, ki doseže 1% točnost. Predlagamo tudi možno implementacijo na spletni strani z namenom izboljšanja poslovnih rezultatov.

Ključne besede: priporočilni sistem, vzporedno računanje, strojno učenje, velike podatkovne množice, matrična faktorizacija.

Abstract

Recommender systems are present almost everywhere on the web and can be the key to potentially improved business results. In this thesis we develop a production-ready recommender system for a website that offers eco-sustainable accommodations, that meet certain requirements (e.g. usage of solar energy, water filtering and reuse, waste recycling etc.). First we examine crucial big data technologies and some of the cloud-based machine learning platforms. We proceed to choose the best platform and use it to collect data and develop a recommender system, which returns predictions for a user, based on a matrix factorization algorithm (*Alternating Least Squares, ALS*). It also returns similar items based on Jaccard similarity and euclidian distance. We conclude with system evaluation by using *Precision@k* statistical measure. The evaluation results have shown 19% precision accuracy, which greatly exceeds the results of random recommendation that achieves 1% precision accuracy. We also propose a potential website implementation with the intention of improving business results.

Keywords: recommender system, parallel computing, machine learning, big data, matrix factorization.

Poglavje 1

Uvod

V današnji internetni dobi uporabniki pričakujejo takojšnje in točne odgovore na poljubne poizvedbe. Podjetja se zato osredotočajo na ustvarjanje informacij, ki so ciljane na točno določenega uporabnika, saj je to nujno potrebno za poslovni uspeh. Primer tega so ciljani oglasi in personalizirani priporočilni sistemi, ki se pogosto pojavljajo na spletnih straneh. Nastanitvena dejavnost je poleg spletnih trgovin najbolj izpostavljena panoga spletnega trženja in posledično zahteva velika vlaganja za pridobitev konkurenčne prednosti.

Ljudje morajo pogosto sprejemati odločitve brez lastnih izkušenj ali znanja o alternativnih možnostih. V vsakodnevnem življenju se v takem primeru obrnemo na priporočila in mnenja drugih ljudi. V preteklosti je bilo izražanje priporočil vedno eksplicitno [20]. Sodobni priporočilni sistemi pa iz množice implicitnih podatkov generirajo in ponujajo priporočila, ki so personalizirana in osnovana na implicitnih dejanjih ali pa eksplicitnih mnenjih in ocenah milijonov drugih uporabnikov.

Priporočilni sistemi in dinamično uravnavanje cen so ključnega pomena, saj lahko pomenijo znatno izboljšanje poslovnih rezultatov podjetij. Podjetja, kot so TripAdvisor, Booking.com in Agoda.com, implementirajo priporočila skozi celotno uporabniško izkušnjo, vse od začetka, ko uporabnik prvič pride na spletno stran ter so mu priporočene najbolj priljubljene destinacije in nastanitve, do specifičnega iskanja, ko uporabnik že povsem speci-

ficira svoje želje in mu sistem ponudi nastanitve, podobne njegovim iskalnim zahtevam.

Soočamo se z izjemno rastjo uporabnikov in uporabe računalniških tehnologij (na področjih ekonomije, medicine, socialnih omrežij, medijev, ...) in svetovnega spleta, kar obenem pomeni, da generiramo več podatkov kot kadarkoli prej. Ti podatki so večinoma kompleksni in nestrukturirani, kar pomeni, da ne ustrezajo nobeni vnaprej definirani obliki. Obenem se generirajo izjemno hitro in v zelo velikih količinah. Takšnih podatkov ne moremo obdelati z uporabo tradicionalnih metod obdelave podatkov. Skupno jih imenujemo **velike podatkovne množice** (angl. *big data*).

Izzivi, ki jih takšni podatki ustvarjajo, so kako shraniti veliko količino nestrukturiranih podatkov in kako obdelati te podatke ter posledično najti in pridobiti znanje iz takšnih podatkov.

Rešitve, ki odgovarjajo na ta vprašanja, pomagajo organizacijam odkriti novo znanje in pridobiti dodano vrednost iz njihovih podatkov. Ena najpogostejših in v zadnjem času najbolj zanimivih aplikacij tako pridobljenega znanja so že omenjeni priporočilni sistemi. Ker imamo na voljo izjemno velike količine podatkov v obliki velikih podatkovnih množic, lahko priporočilni sistemi ustvarijo kvalitetnejše napovedne modele in personalizirana ter kvalitetnejša priporočila kot kdajkoli prej.

1.1 Opis problema

V našem delu se ukvarjamo s priporočilnimi sistemi, konkretno z razvojem priporočilnega sistema za priporočanje nastanitev. Priporočilni sistem razvijemo za spletno stran `http://www.ecobnb.com`, ki ponuja nastanitve, ki dosegajo določene ekološke standarde. Kot ekološko ustrezna nastanitev je definirana vsaka, ki ponuja vsaj pet od desetih zahtevanih storitev. Primeri teh zahtevanih storitev so uporaba sončne energije, filtriranje in ponovna uporaba vode, recikliranje odpadkov itd.

Splošen namen priporočilnih sistemov je ustvariti smiselna priporočila za

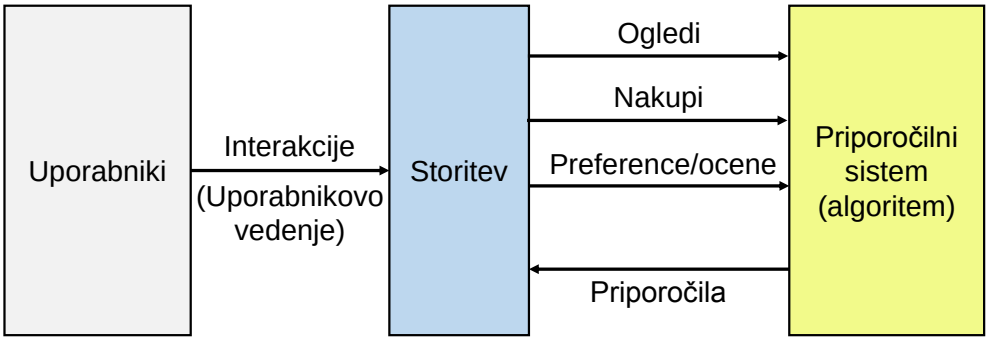
nekega uporabnika glede na njegove pretekle interakcije, kar je shematsko prikazano na sliki 1.1. Entitete, ki jih sistem uporabniku priporoči, imenujemo *priporočilni objekti* (angl. *recommended items*). Poleg priporočanja uporabniku implementiramo tudi priporočanje podobnih priporočilnih objektov na osnovi lastnosti teh objektov. Glavni problem in končni cilj sistema sta priporočiti takšne priporočilne objekte, da bodo uporabniku zanimivi [18]. Razvijemo torej sistem, ki bo uporabniku ponudil personaliziran seznam nastanitev, za katere oceni, da bodo uporabniku zanimive in obenem omogoča tudi priporočanje podobnih nastanitev, za katere oceni da so po določenih lastnostih podobne trenutno opazovani nastanitvi. Med temi lastnostmi najdemo tudi prej omenjene zahtevane ekološke standarde. Gre torej za širšo množico lastnosti nastanitev, ki vključuje ekološke standarde (npr. uporaba sončne energije), tip nastanitve (npr. hostel, kmetija, hotel itd.), tematike (npr. družina, pari, mladi itd.) in storitve (npr. bazen, internet, zajtrk itd.).

Jedro problema, s katerim se soočamo, predstavlja priporočilni algoritem. Ta problem najprej rešujemo z uporabo algoritma matrične faktorizacije. Za implementacijo matrične faktorizacije uporabimo algoritem *Alternating Least Squares (ALS)*. Uporaba tega algoritma je smiselna skupaj z uporabo porazdeljenih tehnologij za shranjevanje in obdelavo velikih podatkovnih množic, saj se algoritem lahko zelo uspešno izvaja vzporedno na več vozliščih. Poleg tega razvijemo tudi lasten algoritem z implementacijo Jaccardove podobnosti in evklidske razdalje, ki izračuna skupno oceno podobnosti med dvema priporočilnima objektoma.

1.2 Namen naloge in rezultati

Namen naloge je razviti priporočilni sistem za mednarodno turistično spletno stran, ki je osnovan na porazdeljenih in skalabilnih tehnologijah velikih podatkovnih množic. Za algoritem uporabimo matrično faktorizacijo ter lasten algoritem na osnovi Jaccardove podobnosti in evklidske razdalje.

Priporočilni sistem bo uporabniku na voljo kot storitev (angl. *"Recom-*



Slika 1.1: Shematski prikaz splošnega delovanja priporočilnih sistemov.

mendation as a Service"), kar pomeni, da bo uporabnik lahko sistem preprosto vključil v lastno rešitev, medtem ko bo sistem postavljen in se bo izvajal na oddaljeni strojni opremi v oblaku.

Priporočilni sistem bo ponujal dve glavni funkcionalnosti:

1. priporočanje nastanitev, za katere je najbolj verjetno, da bi uporabnika zanimale (samo za uporabnike z minimalno potrebno preteklo interakcijo s spletno stranjo),

2. priporočanje podobnih nastanitev (similar items) podani nastanitvi.

Na koncu ocenimo uspešnost razvitega sistema z uporabo statistične mere *Precision@k*.

1.3 Struktura naloge

V prvem delu diplomske naloge (poglavje 1) opisujemo problem in namen našega dela ter končni pričakovani izdelek. Naše delo podrobno opišemo in ga povežemo z ustreznimi publikacijami.

V naslednjem delu (poglavje 2) podajamo grob pregled področja priporočilnih sistemov. Potem podrobneje analiziramo področje velikih podatkovnih množic ter opišemo vsako izmed tehnologij, ki sestavljajo sklad za

strojno učenje, ki podpira uporabo velikih podatkovnih množic. Nato pregledamo nekatere izmed glavnih ponudnikov strojnega učenja v oblaku in njihove rešitve.

V sledečem poglavju (poglavje 3) naš priporočilni sistem podrobneje razdelamo. Najprej razdelamo arhitekturo, kjer sestavne dele izbranega strežnika za strojno učenje umestimo v celotno visokonivojsko arhitekturo. Za tem opišemo storitev, ki izpostavlja naš sistem navzven in jo uporabljamo za zbiranje podatkov. Na koncu podrobneje opišemo področje priporočilnih sistemov in podrobno analiziramo teoretično ozadje, torej algoritme in konceptualni postopek reševanja našega problema, skupaj s pripadajočimi enačbami.

V zadnjem poglavju (poglavje 4) natančno opišemo vse faze implementacije naše rešitve za priporočanje nastanitev v turizmu, podrobneje opišemo podatke in se osredotočimo na evalvacijo in rezultate ter implementacijo sistema v produkcijskem okolju.

1.4 Projekti in povezane publikacije

Problem, ki ga bomo v diplomskem delu reševali in podrobneje opisali, je del evropskega projekta *"EcoDots"*, pri katerem sodeluje tudi Odsek za inteligentne sisteme (E9) kot del raziskovalnega Instituta "Jožef Stefan". Del projekta, ki ga bomo opisali, obsega razvoj priporočilnega sistema za spletno stran, ki ponuja ekološke nastanitve na območju srednje in JV Evrope.

Prva študija področja velikih podatkov, priporočilnih sistemov in strojnega učenja v oblaku je bila na omenjenem odseku izvedena v letu 2014 in sicer v obliki projekta *Analiza nakupovalnih navad kupcev v spletnih trgovinah.* Tekom tega projekta smo se v razvojni skupini prvič seznanili s področjem velikih podatkovnih množic, analizirali ponudnike strojnega učenja v oblaku, ki so bili takrat v zgodnji fazi razvoja, ter področjem priporočilnih sistemov. Projekt se je zaključil z objavo delovnega poročila in članka na konferenci Inteligentni sistemi [23].

Kasneje v letu 2014 in 2015 smo znanje, pridobljeno na prejšnjem pro-

jektu, uporabili za razvoj priporočilnega sistema v omenjenem projektu *"Eco-Dots"*.

Celotna analiza strojnega učenja v oblaku in področja velikih podatkov se je vseskozi izvajala pod okriljem projekta *Raziskovalci na začetku kariere*. Projekt se je zaključil v letu 2015 s pripravo obsežnega delovnega poročila z naslovom *Raziskave prilagodljivih domenskih napovednih modelov*.

Avtor in tematika. Avtor tega diplomskega dela sem kot del razvojne ekipe ter soavtor sodeloval pri vseh naštetih projektih in publikacijah. Tematika se kot plod mojega dela na določenih mestih delno prekriva, vendar se obenem znatno razlikuje v poudarkih in podrobnostih. Diplomsko delo je povsem samostojno, unikatno in v celoti ustvarjeno kot individualno delo avtorja in mentorja.

Poglavje 2

Pregled področja

2.1 Priporočilni sistemi in algoritmi

Priporočilni sistemi so danes vsenavzoči in spreminjajo način, na katerega uporabniki najdejo informacije, produkte, druge ljudi itd. Ti sistemi analizirajo vzorce vedenja uporabnikov in tako predvidijo, kaj bi nek uporabnik preferiral iz množice predmetov, s katerimi prej še ni imel interakcije.

Cilj priporočilnega sistema je generirati smiselna priporočila predmetov ali izdelkov neki množici uporabnikov. Amazonov priporočilni sistem za knjige in Netflixov priporočilni sistem za filme sta primera najnaprednejših in najuspešnejših industrijsko močnih priporočilnih sistemov [15, 29]. Omenjena priporočilna sistema sta pripravljena na streženje velikemu številu uporabnikov v realnem času in se tako uspešno soočata z nekaterimi izzivi velikih podatkovnih množic [18].

Načrtovanje in razvoj takšnega sistema sta v veliki meri odvisna od domene in specifičnih karakteristik podatkov, ki so na voljo. Pri omenjenih sistemih je na voljo velika množica interakcij med uporabniki in predmeti, saj redno pridobivata ocene uporabnikov za določen predmet. Te ocene so cela števila na lestvici od 1 (uporabniku predmet ni všeč) do 5 (uporabniku je predmet zelo všeč). Takšni podatki vsebujejo informacijo o kvaliteti interakcij med uporabniki in predmeti ter so zelo ustrezni za učenje in po-

sledično generiranje priporočil. Poleg teh imata sistema na voljo še dodatne podatke o uporabnikih (npr. demografski podatki) in predmetih (npr. opisi, ključne besede itd.), ki jih lahko uporabita kot dodatne parametre pri učenju in sklepanju [18]. Tekmovanje "Netflix prize" je spodbudilo izjemen razvoj na področju priporočilnih sistemov, kjer so se uveljavile predvsem metode za matrično faktorizacijo [12, 3, 11]. Te metode izračunajo približke neznanih preferenčnih vrednosti z uporabo matričnega razcepa in minimizacije enačb z metodo gradientnega spusta [12, 27]. S porastom porazdeljenih sistemov in pojavom velikih podatkovnih množic pa gre smer razvoja v vzporedne algoritme [24], katerih primer je metoda Alternating Least Squares (ALS) [12, 28, 25].

Priporočilni sistemi se razlikujejo v načinu, na katerega analizirajo podatke in se posledično naučijo in sklepajo na povezave med uporabniki in predmeti ter najdejo najboljše pare *uporabnik-predmet* [18].

V splošnem poznamo več vrst priporočilnih sistemov. Večino izmed njih lahko klasificiramo v eno izmed dveh glavnih velikih skupin [2, 19, 1, 10, 14].

2.1.1 Skupinsko filtriranje (angl. collaborative filtering).

Gre za najbolj pogosto in najuspešnejšo vrsto priporočilnih sistemov. Ideja algoritma je generirati priporočilo na osnovi mnenj oziroma ocen drugih uporabnikov, ki so imeli v preteklosti podobne preference kot naš uporabnik [19, 1, 10, 14]. Takšen sistem zahteva uporabniško zgodovino in se uporablja predvsem za priporočanje specifičnemu uporabniku – personalizirana priporočila.

Modeli s skritimi spremenljivkami. V zadnjem času so se uveljavili algoritmi, ki iz učnih podatkov zgradijo model, na podlagi katerega lahko vračajo priporočila poljubnemu uporabniku. Takšen model se največkrat zgradi z uporabo matrične faktorizacije. Izračun neznanih spremenljivk v matrikah razcepa se lahko izvede z metodo stohastičnega gradientnega spusta (SGD) ali pa metodo Alternating Least Squares (ALS) [12, 24, 27, 28, 25].

Takšen uporabljen postopek podrobno opišemo v poglavju 3.

2.1.2 Vsebinsko filtriranje (angl. content-based filtering).

Govorimo o drugačni vrsti sistemov, ki se osredotoča na primerjavo opisov predmetov, s katerimi je uporabnik imel interakcijo, ter opisov predmetov, s katerimi ta isti uporabnik še ni imel interakcije in jih želimo priporočiti [2, 18, 1, 14]. Tovrsten sistem ne potrebuje uporabniške zgodovine, ampak samo opise oziroma lastnosti predmetov.

Jaccardov koeficient in evklidska razdalja. Pri vsebinskem filtriranju se med profili priporočilnih objektov računa podobnost. Največkrat se uporabi kosinusna razdalja, v primeru binarnih atributov podatkov pa je smiselna uporaba Jaccardovega koeficienta, ki meri podobnost med dvema končnima množicama vzorcev. Z georafskimi koordinatami je smiselno uporabiti evklidsko razdaljo [15], ki meri oddaljenost med dvema točkama, tipično v dvodimenzionalni evklidski ravnini. Definicije podobnosti in postopek izračuna vsake izmed njih podrobno predstavimo v poglavju 3. Predstavimo tudi predlog postopka združevanja različnih mer podobnosti.

2.1.3 Hibridni pristopi.

Obstajajo tudi druge vrste priporočilnih sistemov, kot so *sistemi na osnovi populacije* (angl. *demographic-based*) in *sodelovanje prek vsebine* (angl. *collaboration via content*), ki so pogosto združena oblika ali hibrid zgoraj omenjenih. Ti sistemi imajo potencial, vendar so bistveno manj pogosti in ne igrajo vidnejše vloge v primerjavi s prvima dvema skupinama, saj zahtevajo zelo specifičen razvoj glede na problemsko domeno in podatke.

2.2 Tehnologije velikih podatkovnih množic

Smiselna analogija za opis tega, kaj so velike podatkovne množice (angl. *big data*) in kakšne možnosti odpirajo, se ponuja v iskanju oziroma rudarjenju zlata. V preteklosti so rudarji zlahka opazili kosce ali žile zlata, saj so bile opazne že na površju. Predpostavimo, da so *z informacijami bogati podatki* zlato. Njihovo vrednost zlahka prepoznamo in vlagamo vire v luščenje informacij. Vendar pa obstaja možnost, da se zlato nahaja tudi drugje, morda v bližini ali pa daleč, a ni zlahka opazno. Potrebnih pa bi bilo ogromno resursov, da bi pregledali vso širšo okolico, brez zagotovil za najdbe. Enako velja za podatke v podatkovnih skladiščih. Ti podatki so pomembni, imajo vrednost in jim zaupamo ter vlagamo resurse v njihovo obdelavo in analizo [30].

Danes pa rudarji delajo drugače. Nove tehnologije omogočajo preiskovanje ogromnih količin zemlje (*z informacijami revni podatki*), da najdejo skoraj nevidne sledi zlata (*z informacijami bogati podatki*). Z drugimi besedami, v veliki količini odpadnega materiala lahko s pravo opremo najdemo tudi veliko zlata. Enako velja za današnje podatke. Na voljo jih je več kot kdajkoli prej in na voljo so tudi orodja za odkrivanje zakonitosti v velikih podatkovnih množicah (angl. *big data*) [30].

Trije V-ji velikih podatkovnih množic. Trije izzivi, v angleščini pogosto imenovani *"Three Vs of big data (3Vs)"*, so obenem tudi tri dimenzije, ki skupaj najpogosteje formalno definirajo velike podatkovne množice. To so:

1. velikost (**V**olume),
2. raznolikost (**V**ariety),
3. hitrost (**V**elocity).

Velikost (Volume). Količina podatkov, ki jih velika podjetja (Twitter, Facebook, Google itd.) hranijo, je eksplodirala. Omenjena podjetja generirajo terabajte podatkov vsakodnevno. Posledično je sama velikost podatkov

postala izziv, saj lahko podjetja analizirajo čedalje manjši delež vseh podatkov, ki jih hranijo [31].

Raznolikost (Variety). Zaradi različnih načinov zbiranja podatkov so ti postali kompleksni, saj niso več le relacijski, marveč tudi surovi, nestrukturirani, multimedijski itd. Vse te oblike podatkov povečujejo raznolikost. Več kot 80% današnjih podatkov je nestrukturiranih [31].

Hitrost (Velocity). Danes podatki niso več statični, ampak se vseskozi spreminjajo. Tradicionalno hitrost pomeni, kako hitro podatki prihajajo ter kako hitro morajo biti obdelati in shranjeni. Danes pa lahko govorimo o toku podatkov v realnem času [31].

2.2.1 Analiza tehnologij

Tehnološke rešitve, ki ponujajo implementacije odgovorov na izzive velikih podatkov, so večinoma odprtokodne in splošno dostopne. V nadaljevanju pregledamo in v grobem opišemo tehnologije, uporabljene v našem sistemu. Opisi posameznih komponent si sledijo v smiselnem vrstnem redu tako, kot si sledijo v izbranem strežniku za strojno učenje `Prediction.IO` od spodaj navzgor (angl. *bottom-up approach*).

Apache Hadoop

Apache Hadoop je odprtokododno ogrodje (angl. *open source framework*) za pisanje in poganjanje porazdeljenih aplikacij, ki obdelujejo velike količine podatkov [13]. Ključne lastnosti, ki definirajo Hadoop, so [13]:

1. **Dostopnost** - Hadoop se lahko uporablja na gručah, sestavljenih iz velikega števila poceni računalnikov, ali pa na oblačnih storitvah za računanje, kakršne na primer ponuja Amazon (Amazon Web Services, AWS).

2. **Robustnost** - Zaradi potencialne uporabe na velikem številu poceni računalnikov je Hadoop zasnovan s predpostavko pogostih okvar strojne opreme. Posledično je zelo odporen na takšne okvare in zagotavlja visoko stabilnost.

3. **Skalabilnost** - Hadoop se stopnjuje linearno za obdelavo velikih količin podatkov z dodajanjem vozlišč v gručo.

4. **Preprostost** - Hadoop omogoča uporabnikom, da zelo hitro spišejo učinkovito kodo za vzporedno računanje.

Apache HBase

HBase se uvršča med stolpično orientirane podatkovne baze, saj podatke shranjuje na disk v stolpično orientiranem formatu. Pomembna prednost baze HBase je v tem, da omogoča zelo hiter dostop do določene celice ali sekvence celic z uporabo ključev [5].

V splošnem se HBase uporablja za shranjevanje in pridobivanje podatkov z uporabo naključnega dostopa (angl. *random read*), kar pomeni, da lahko podatke zapisujemo na poljuben način in jih hitro preberemo iz baze, kadarkoli jih potrebujemo. Podpira shranjevanje strukturiranih in nestrukturiranih podatkov (z določeno velikostno omejitvijo). Tipi podatkov niso pomembni, kar pomeni, da lahko uporabimo dinamičen in fleksibilen podatkovni model, ki ne omejuje oblike in tipa podatkov za shranjevanje, kot je to tipično za relacijske sheme.

HBase je ustvarjena za uporabo na gruči računalnikov in ne samo na enem. To gručo lahko sestavlja večje število poceni računalnikov. Vsako vozlišče prispeva delež shrambe, pomnilnika in računske moči. To zagotavlja veliko fleksibilnost in odpornost na potencialno okvaro posameznega računalnika, saj nobeno vozlišče ni unikatno, temveč so kopije manjših enot podatkov prisotne na večih vozliščih [4].

Apache Spark

Spark razširja popularni model Hadoop MapReduce z namenom podpore več vrst računskih operacij, kot so interaktivne poizvedbe in obdelava toka podatkov v realnem času. Pri velikih količinah podatkov je pomembna hitrost obdelave. Spark omogoča hranjenje vmesnih rezultatov med obdelavo podatkov znotraj pomnilnika (RAM), kar je bistvena prednost in prinese znatne pohitritve. Obenem je tudi učinkovitejši kot MapReduce za kompleksne aplikacije, ki se izvajajo na navideznem pomnilniku (HDD) [7].

Spark MLlib. Gre za knjižnico, ki vsebuje pogoste algoritme za strojno učenje, kot so na primer klasifikacija, regresija, razvrščanje v skupine oz. gručenje, skupinsko filtriranje itd.

Za to diplomsko delo je ključna podpora MLlib za algoritme skupinskega filtriranja (CF), ki se najpogosteje uporabljajo v priporočilnih sistemih.

Elasticsearch

Elasticsearch je odprtokodni iskalni strežnik, ki uporablja knjižnico Apache Lucene, vendar zakriva kompleksne podrobnosti implementacije in ponuja uporabniku preprost programski vmesnik za uporabo. Podatke shranjuje v obliki dokumentov JSON, vendar brez vnaprej definirane sheme [6].

2.3 Strojno učenje v oblaku

Skupaj z razvojem opisanih tehnologij, ki podpirajo shranjevanje in obdelavo velikih količin nestrukturiranih podatkov, se je razvijala tudi ideja, kako uporabiti te tehnologije za strojno učenje in to na preprost način ponuditi razvijalcem in uporabnikom, ki niso eksperti v strojnem učenju in podatkovni znanosti (angl. *data science*).

V zadnjih petih letih se je na trgu pojavila množica ponudnikov t.i. strojnega učenja kot storitev (angl. *Machine Learning as a Service, MLaaS)* oziroma strojnega učenja v oblaku (angl. *cloud-based machine learning*). Uve-

ljavil se je tudi izraz "platforma kot storitev" (angl. *Platform as a Service, PaaS*), ki nakazuje celostno arhitekturno podporo, od podatkovne shrambe, prek algoritmov do programskih vmesnikov za komunikacijo z odjemalcem. Večini je skupno to, da nudijo strojno in programsko opremo v oblaku ter ju uporabniku izpostavijo samo preko programskih vmesnikov kot storitev REST. Na ta način se uporabniku ni potrebno ukvarjati z namestitvijo in vzdrževanjem, temveč lahko prenese le svoje podatke na oddaljen sistem, kjer se ti podatki obdelajo. Sistem se na njih nauči in zgradi model ter nato vrača rezultate.

2.3.1 Primerjava platform MLaaS

V prvi fazi izbora ustreznega podpornega sistema MLaaS za naš problem analiziramo in med seboj primerjamo več ponudnikov. Osredotočimo se na pregled objektivnih lastnosti platforme, kjer smo največji poudarek namenili odprtosti rešitve, ki omogoča razumevanje in možnost lastne modifikacije, ter algoritmom za strojno učenje. Primerjava je vidna v tabeli 2.1.

Platforma	Odprtost sistema	Jeziki programskih vmesnikov	Podprti algoritmi	Predloge aplikacij strojnega učenja
Google Prediction API	Black-box	Go, Java, JavaScript, .NET, Node.js, PHP, Python, Ruby	Klasifikacija, regresija	Ni podprto
BigML	Black-box	Python, Node.js, Java, Bash, C#, R, Ruby, PHP	Odločitvena drevesa	Ni podprto
Azure ML	Black-box	Python, C#, R	Odločitvena drevesa, nevronske mreže, klasifikacija, regresija, gručenje itd.	Implementacije algoritmov: https://gallery.azureml.net
Amazon ML	Black-box	AWS API	Klasifikacija, regresija	Ni podprto
Prediction.IO	White-box	Java, Python, PHP, Ruby, Node.js, C#, Swift	SVM, klasifikacija, regresija, gručenje, odločitvena drevesa, skupinsko filtriranje (CF) itd.	Implementacije algoritmov: https://templates.prediction.io/

Tabela 2.1: Primerjava lastnosti ponudnikov MLaaS, ki so zanimive za razvijalce.

`Prediction.IO` je edina izmed primerjanih platform MLaaS, ki nudi popoln vpogled v programsko kodo in s tem omogoča podrobno razumevanje vseh delov sistema. Ključno omogoča tudi poljubno spreminjanje programske kode in s tem implementacijo lastnih algoritmov strojnega učenja (ali spremembo obstoječih) ter lastnega shranjevanja podatkov. Dodana vrednost za razvijalca so implementacije določenih algoritmov strojnega učenja v obliki predlog. Zaradi teh prednosti in podrobne dokumentacije za razvoj priporočilnega sistema izberemo platformo Prediction.IO, ki jo podrobneje opišemo v nadaljevanju.

2.4 Obstoječe rešitve

Rešitve, ki ponujajo celoten sklad s podporo shranjevanju podatkov, strojnemu učenju in algoritmom ter komunikaciji z odjemalci, so zelo nedavne in jih primerjamo v prejšnjem razdelku. Obstaja pa množica že uveljavljenih rešitev v obliki knjižnic, ki se osredotočajo na algoritme priporočanja.

Omejimo se na rešitve, ki so odprtokodne in prosto dostopne.

Apache Mahout. Gre za paket knjižnic za strojno učenje, ki so skalabilne in podpirajo široko paleto algoritmov za strojno učenje, vključno z algoritmi skupinskega filtriranja. Matrična faktorizacija je podprta in implementirana na računskem modelu Hadoop MapReduce.

MyMediaLite. Je knjižnica, razvita v programskem jeziku C#, ki implementira množico algoritmov skupinskega filtriranja. Podpira tudi matrično faktorizacijo, ki je implementirana na platformi. NET.

LibRec. Zelo nedavna, knjižnica razvita v programskem jeziku Java. Podpira veliko množico algoritmov skupinskega filtriranja, vključno z matrično faktorizacijo in algoritmom ALS.

Apache Spark MLlib. Knjižnica z mnogimi algoritmi skupinskega filtriranja. Je tesno povezana s knjižnico Apache Mahout, saj se nekatere implementacije algoritmov prekrivajo. Bistvena razlika je v računskem modelu, saj je MLlib implementirana nad modelom Apache Spark, ki je za določene naloge bistveno hitrejši od modela MapReduce, ki ga uporablja Mahout.

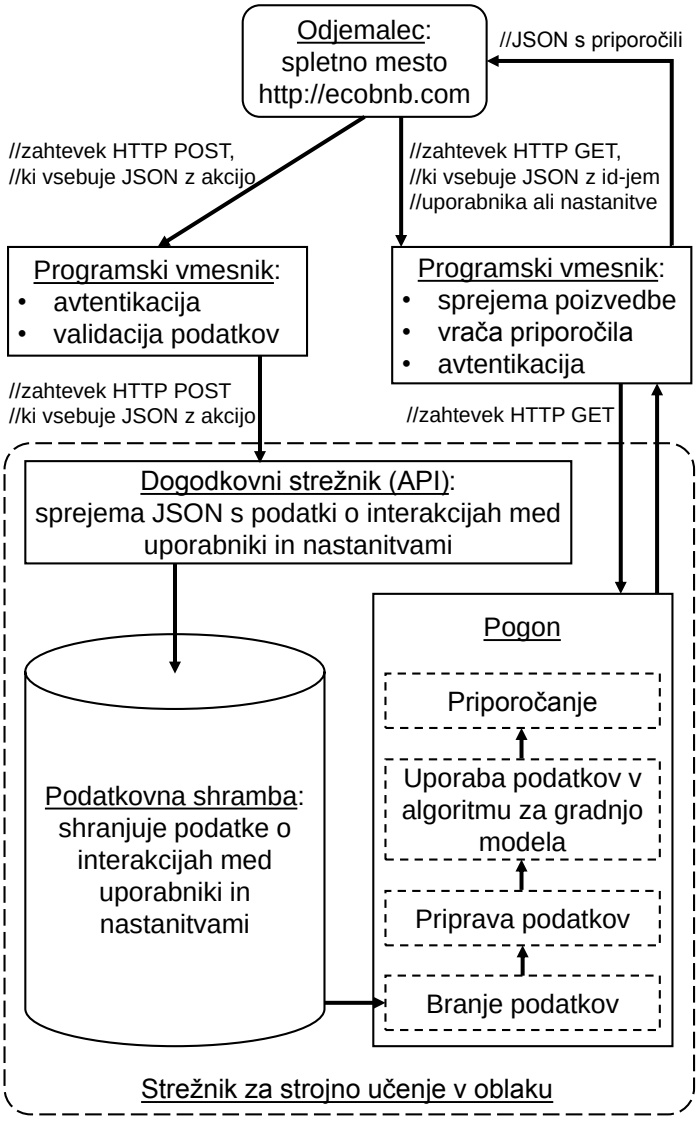
Razlika in prednost našega sistema. Naš priporočilni sistem se razlikuje od ostalih rešitev predvsem v tem, da podatke zbiramo v realnem času, jih sami oblikujemo in glede na to obliko prilagodimo algoritem matrične faktorizacije, da lahko uporabi zbrane podatke. Konkretno implicitne akcije smiselno pretvorimo v izkaz različnih preferenc, ki jih algoritem nato uporabi za izgradnjo modela. Dodatno implementiramo funkcionalnost priporočanja podobnih nastanitev, kar dosežemo z uporabo algoritma Jaccardove podobnosti in evklidske razdalje, ki ju smiselno združimo v skupno vrednost podobnosti.

Znotraj enega sistema razvijemo priporočanje za uporabnika na osnovi matrične faktorizacije ter priporočanje podobnih nastanitev na osnovi lastnosti posameznih nastanitev. Implementiramo algoritem skupinskega filtriranja in algoritem na osnovi vsebinskega filtriranja v enotno storitev, kar v nobeni izmed ostalih rešitev ni omogočeno. Ostale rešitve se večinoma osredotočajo samo na skupinsko filtriranje in vedno zahtevajo ločene podatke in ločen razvoj algoritmov iz različnih skupin priporočilnih algoritmov. Naša rešitev ponuja prednost, saj se lahko uporabi tako v okolju, kjer so na voljo zgolj zgodovinske interakcije med uporabniki in priporočilnimi objekti, kot v okolju, kjer imamo na voljo le lastnosti priporočilnega objekta.

Poglavje 3

Opis predlagane rešitve

Priporočilni sistemi so danes skorajda obvezna funkcionalnost aplikacij, ki se ukvarjajo s trženjem prek spleta (angl. *e-Commerce*). V preteklosti so bili priporočilni sistemi redkost, saj je implementacija takšnega sistema zahtevala veliko resursov in specifičnih znanj. Zaradi tega so bili takšni sistemi v domeni velikih podjetij, kot so Amazon, Ebay itd.



Slika 3.1: Načrt priporočilnega sistema.

V naslednjem razdelku podrobneje opišemo načrt priporočilnega sistema, prikazanega na skici 3.1, ki ustreza zgradbi strežnika za strojno učenje Prediction.IO in prikazuje interakcijo med odjemalcem in priporočilnim sistemom, tako v fazi zbiranja podatkov kot v fazi pridobivanja priporočil.

3.1 Zgradba strežnika za strojno učenje

Strežnik Prediction.IO za strojno učenje v oblaku, ki smo ga uporabili za naš projekt, v grobem sestoji iz naslednjih treh komponent:

- **PredictionIO platforma (angl. PredictionIO platform).** Odprtokoden sklad, sestavljen iz opisanih porazdeljenih in skalabilnih tehnologij, ki uspešno rešujejo izzive velikih podatkovnih množic. Sklad omogoča izgradnjo pogona in evalvacije ter postavitev pogona kot storitev v oblaku.

- **Dogodkovni strežnik (angl. Event Server).** Programski vmesnik za zbiranje podatkov v realnem času, ki zagotavlja enotno obliko zbranih podatkov, ne glede na izvorno platformo.

- **Galerija predlog (angl. Template Gallery).** Spletno mesto, ki ponuja vnaprej razvite splošne pogone, imenovane predloge. Predloge aplicirajo algoritme strojnega učenja na realnih problemih in se lahko prilagodijo lastni problemski domeni.

Naštete komponente smo predvideli v našem načrtu in jih lahko prepoznamo na sliki 3.1.

3.1.1 Arhitektura MVC za strojno učenje v oblaku

MVC (angl. *model-view-controller*) je uveljavljen tip arhitekture, ki je najpogosteje v uporabi pri razvoju spletnih aplikacij. Platforme MVC za razvoj spletnih aplikacij, kot so Django, Ruby on Rails, Spring MVC, ASP.net itd., so razširjene in priljubljene.

Zaradi prednosti arhitekture MVC bi bilo podoben pristop smotrno implementirati in uporabljati tudi na področju strojnega učenja v oblaku. Prediction.IO predstavlja enega izmed prvih poskusov uveljavitve takšnega pristopa na tem področju. Arhitektura MVC se kaže v modularni zasnovi posameznega pogona, ki jo opiše kratica **DASE**:

- **D** – Data Source/Data Preparator. Ta del je odgovoren za branje podatkov iz podatkovne shrambe in potencialno predprocesiranje oziroma pripravo podatkov za uporabo v algoritmu.
- **A** – Algorithm. Tu se učni podatki v točno določeni obliki uporabijo v algoritmu, katerega rezultat je napovedni model.
- **S** – Serving. Sprejema poizvedbe, jih posreduje napovednemu modelu in vrača rezultate. Nujno potrebna komponenta za komunikacijo med odjemalcem in pogonom.
- **E** – Evaluator. Opcijska komponenta, kjer se na poljuben način z uporabo neke statistične mere oceni relativna kvaliteta algoritma.

Takšna arhitektura na nivoju aplikacije omogoča to, da lahko razvijalec zamenja le algoritem, ostali deli pogona pa ostanejo enaki in bo aplikacija še vedno delovala. Jedrni del vsake komponente torej ostane enak, kar omogoča večkratno uporabo iste kode in zmanjša zahtevno programiranje.

Dogodkovni strežnik

Prva ključna komponenta znotraj izbranega strežnika za strojno učenje Prediction.IO je dogodkovni strežnik (angl. *event server*). Gre za programski vmesnik, ki sprejema učne podatke s strani odjemalcev v realnem času. Sprejemanje podatkov se izvaja v skladu s t.i. dogodkovno paradigmo, kar pomeni, da vsaka enota podatkov predstavlja en dogodek. Na strežniku obstajajo določene omejitve in vnaprej definirani dogodki, ki se uporabljajo za tipične primere podatkov, ki jih ne moremo opisati na dogodkoven način (npr. podatki o uporabnikih in priporočilnih objektih).

Primer enote podatka. Primer preprostega dogodkovnega podatka je: uporabnik *U1* na odjemalcu izvede *ogled* izdelka *I1*. Takšen dogodek vsebuje ključne povezane podatke za hrambo, ki so uporabnik *U1*, priporočilni objekt *I1* in interakcija med njima, v tem primeru akcija *ogled* (angl. *view*).

Primer podatka, ki ga ne moremo opisati na dogodkoven način, je: na odjemalcu smo dodali nov izdelek *I2* v ponudbo. Tega izdelka do sedaj ni bilo na voljo. Takšne podatke, ki tipično opišejo uporabnike ali priporočilne objekte, hranimo v isti shrambi, v skupini dogodkov *$set*, kjer akcija *$set* predstavlja nastavitev lastnosti uporabnika ali priporočilnega objekta, ne pa interakcije.

Podatkovna shramba

Ko je dogodek sprejet, se shrani v podatkovno shrambo (angl. *event data-store*). V strežniku Prediction.IO je privzeto to podatkovna baza Apache HBase, kamor se shrani vsak dogodek v realnem času. Podprte so tudi alternativne možnosti za podatkovno shrambo, kot so MongoDB, MySQL itd.

Pogon

Pogon (angl. *engine*) je neodvisna celostna aplikacija za strojno učenje. Sestavljena je iz vseh komponent DASE oziroma nujno vsaj prvih treh, saj je evalvacija opcijska. Engine združi vse komponente DASE v delujočo aplikacijo, ki se lahko postavi kot storitev.

Pogon izvaja naslednje naloge:

- prebere in uporabi učne podatke v algoritmu za izgradnjo napovednega modela,
- omogoča dostop zunanjim odjemalcem preko spletne storitve,
- sprejema poizvedbe in vrača rezultate, bodisi v realnem času, bodisi v paketu (batch).

3.2 Storitev REST za zbiranje podatkov

Dogodkovni strežnik ponuja že razvit programski vmesnik, ki se preprosto uporablja za zbiranje podatkov. Vendar mu manjkata avtentikacija in avtorizacija, kar je ključna pomankljivost za produkcijske sisteme. V kolikor bi

nekdo želel dogodkovni strežnik direktno uporabiti za zbiranje podatkov, bi to pomenilo izpostavljanje odprtega programskega vmesnika in prosto uporabo komurkoli, kar pa za produkcijsko okolje ni sprejemljivo.

Z namenom odprave te pomankljivosti ter boljšega razhroščevanja in pregleda interakcije med odjemalcem in strežnikom, smo razvili še en dodaten vmesen programski vmesnik, kjer smo poskrbeli za osnovno varnost in ločili razvojno ter produkcijsko okolje.

Tehnologija storitve. Želeli smo uporabiti preprosto in učinkovito ogrodje za razvoj programskega vmesnika, ki po potrebi omogoča nadgradnje za potrebne dodatne funkcionalnosti. Odločili smo se za Python Flask, ki ponuja izjemno modularno zasnovo, kjer je jedro storitve zelo preprosto, vsaka dodatna funkcionalnost pa se lahko zlahka implementira nad tem preprostim jedrom.

3.3 Teoretično ozadje in metodologija

V literaturi se tipično omenjata dve veliki strategiji za priporočilne sisteme [2, 12, 2], ki smo ju že omenili v poglavju 2 in ju bomo podrobneje opisali v nadaljevanju. To sta:

1. vsebinsko filtriranje ali content-based filtering,
2. skupinsko filtriranje ali collaborative filtering.

Vsebinsko filtriranje (angl. content-based filtering). Gre za pristop, ki ustvari profil za vsak priporočilni objekt v sistemu. Takšen profil opisuje priporočilni objekt z nekimi lastnostmi objekta, ki so vnaprej poznane.

Film lahko opišemo z žanrom, ki pripada filmu, igralci, ki v filmu nastopajo, režiserjem, ki je film posnel, itd. Podobno lahko opišemo neko nastanitev s filtri, ki tej nastanitvi pripadajo (npr. internet, TV, kopalnica, bazen itd.), geografskimi koordinatami (tipično zemljepisna širina in dolžina), cenovnim rangom itd.

Takšni profili omogočajo priporočilnemu sistemu najdbo povezav med podobnimi profili, kar implicitno pomeni najdbo podobnih priporočilnih objektov v sistemu. Problem takšnega sistema je pogosto pridobitev podatkov, ki so potrebni za ustvarjanje tovrstnih profilov priporočilnih objektov. Za ustvarjanje profila in opis priporočilnega objekta namreč rabimo podatke iz zunanjih virov (npr. podatkovne baze), ki pa jih ne moremo pridobiti z opazovanjem, ampak so obstoječa lastnost vsakega priporočilnega objekta.

Zelo znan primer takšnega sistema z vsebinskim filtriranjem je Music Genome Project. To je sistem, ki je v uporabi na spletni strani `http://www.pandora.com` in deluje tako, da glasbeni strokovnjaki vsaki pesmi dodelijo stotine različnih glasbenih lastnosti oz. karakteristik. Te lastnosti skupaj sestavljajo unikaten profil vsake pesmi in pomagajo sistemu najti in predvajati pesmi, podobne tisti, ki jo uporabnik trenutno posluša [12].

Skupinsko filtriranje (angl. Collaborative Filtering, CF). Gre za drugačen pristop, ki se zanaša zgolj na zgodovino interakcij med uporabniki in priporočilnimi objekti. To pomeni, da ne potrebuje dodatnih informacij oz. lastnosti priporočilnih objektov za izgradnjo profila, kar poenostavi pridobitev ustreznih podatkov. CF analizira pretekle povezave oz. interakcije med uporabniki in priporočilnimi objekti ter na podlagi takšne zgodovine predvidi najbolj verjetne nove povezave, ki jih vrne kot priporočila.

Modeliranje preteklih interakcij med uporabniki in priporočilnimi objekti je ena glavnih prednosti CF sistemov, saj takšnih interakcij ne moremo modelirati z vsebinskim filtriranjem. Literatura v splošnem navaja sisteme CF kot bolj natančne v primerjavi z vsebinskim filtriranjem [12], vendar imajo tudi znano težavo, imenovano *hladen začetek* (angl. *cold start*). Ta pojav je tipično viden pri novih uporabnikih ali priporočilnih objektih v sistemu, saj sistem nima dovolj ali pa sploh nobenih podatkov o novih vnosih, kar onemogoča smiselno delovanje. Nujna je namreč čim bolj obsežna zgodovina interakcij, ki pa pri novih uporabnikih sploh ne obstaja in zato sistem ne more generirati smiselnih priporočil.

Dve najpogosteje omenjani področji metod CF sta [12]:

1. metode za lokalno učenje iz najbližjih sosedov (angl. *neighborhood methods*),

2. modeli s skritimi spremenljivkami (angl. *latent factor models*).

Metode za lokalno učenje iz najbližjih sosedov se osredotočajo na izračun relacij med uporabniki in priporočilnimi objekti, tipično z uporabo mere podobnosti, medtem ko modeli s skritimi spremenljivkami poskušajo opisati relacije z ustvarjanjem numeričnih faktorjev. Obe metodi podrobneje opišemo v razdelkih 3.3.1 in 3.3.2.

Umestitev problema. Problem, s katerim se v diplomskem delu soočamo, bi lahko uvrstili v oba opisana razreda. Po eni strani se uvršča v razred skupinskega filtriranja, saj se zanaša na uporabo preteklih interakcij med uporabniki in priporočilnimi objekti. Na drugi strani pa lahko z ustreznim pridobivanjem dodatnih podatkov in predprocesiranjem problem uvrstimo tudi v razred vsebinskega filtriranja.

Ker so prevladujoči podatki interakcije med uporabniki in priporočilnimi objekti in ker smo se osredotočili na zbiranje teh podatkov v realnem času, naš problem definiramo kot problem skupinskega filtriranja.

3.3.1 Metode najbližjih sosedov

Metode najbližjih sosedov se naprej delijo na dva klasična pristopa, ki se razlikujeta glede na uporabo podatkov [12].

Pristop na osnovi uporabnikov (angl. user-based). Prvi pristop je orientiran na uporabnike (angl. *user-based*). Ta pristop uporablja podatke o uporabnikih, in sicer tako, da poišče druge uporabnike, ki so podobno ocenili iste priporočilne objekte, kot jih je ocenil naš izbrani uporabnik. Ko najde takšne uporabnike, pogleda, katere druge priporočilne objekte so ti podobni

uporabniki visoko ocenili, in jih priporoči našemu izbranemu uporabniku. Ta postopek lahko opišemo v dveh korakih:

1. Poišči uporabnike, ki iste priporočilne objekte ocenjujejo podobno kot naš izbrani uporabnik. Najdi torej uporabnike, ki imajo podobne preferenčne vzorce kot uporabnik, za katerega generiramo priporočila.

2. Uporabi preference teh podobnih uporabnikov, ki so bili najdeni v koraku 1, in predlagaj tiste priporočilne objekte, ki so jih ti podobni uporabniki visoko ocenili. Opcijsko: med temi priporočilnimi objekti izberi samo tiste, s katerimi naš uporabnik še ni imel interakcije.

Ta koraka zapišemo v psevdokodi:

Algoritem 1: Priporočanje s pristopom na osnovi uporabnikov.

```
1 function naOsnoviUporabnikov (U, I)
  Input  : Množica uporabnikov U in množica priporočilnih objektov I1
  Output: Najvišje rangirani priporočilni objekti I2
2 foreach priporočilni objekt i za katerega uporabnik u še nima
  preference do
3     foreach uporabnik v, ki ima preferenco za priporočilni objekt i do
4         Izračunaj podobnost s med uporabnikoma u in v;
5         Izračunaj tekoče povprečje preferenc uporabnika v za
          priporočilni objekt i, uteženo s podobnostjo s;
6         Vrni priporočilne objekte I2, ki so rangirani po tem uteženem
          povprečju;
7     end
8 end
```

Takšni priporočilni sistemi so bili v preteklosti popularni in so se pojavili med prvimi, saj so najbolj intuitivni, vendar se soočajo s kar nekaj težavami:

- Slabo se obnesejo v sistemih, kjer je v podatkih veliko priporočilnih objektov, a malo izkazov preferenc. To je pogosta situacija v današnjih

velikih spletnih trgovinah, kjer je na voljo milijone izdelkov, vendar mnogi izmed njih nimajo veliko ali pa sploh nobene ocene.

- Velika poraba resursov in časovna zahtevnost za izračun podobnosti med vsemi pari uporabnikov.

- Uporabniški profili se pogosto in hitro spreminjajo, kar posledično zahteva pogosto osveževanje modela za priporočila. Konkretno to pomeni ponovno izgradnjo oz. učenje na spremenjenih podatkih.

Pristop na osnovi priporočilnih objektov (angl. item-based). Naštete težave za sisteme, kjer je več uporabnikov kot priporočilnih objektov in izkazov preferenc, rešuje pristop, ki temelji na priporočilnih objektih (angl. *item-based*). Ta pristop se razlikuje od prejšnjega v tem, da najprej izračuna podobnosti med vsemi priporočilnimi objekti, nato pa, ko želimo vrniti napoved, sistem pogleda najvišje ocenjene priporočilne objekte izbranega uporabnika in ustvari utežen seznam najbolj podobnih priporočilnih objektov. Prej izračunano podobnost torej utežimo z znanimi ocenami in povprečimo. Na koncu priporočimo priporočilne objekte z najvišjimi izračunanimi vrednostmi [21, 22]. Ta postopek lahko povzamemo v dveh korakih:

1. Z uporabo ene izmed znanih funkcij za izračun podobnosti izračunaj podobnost med vsemi pari priporočilnih objektov. Funkcijo izberi glede na tip podatkov (npr. evklidska razdalja, kosinusna razdalja, Manhattanska razdalja, Jaccardov koeficient, Pearsonov koeficient, log-likelihood ratio itd.).

2. Glede na izračunane podobnosti ustvari seznam vrednosti, ki so povprečene utežene podobnosti neznanih priporočilnih objektov in priporočilnih objektov, ki jih je izbrani uporabnik ocenil. Uteži so znane ocene našega uporabnika, podobnost pa je izračunana v koraku 1. Priporoči izdelke z najvišjimi izračunanimi vrednostmi.

Koraka zapišemo v psevdokodi:

Algoritem 2: Priporočanje s pristopom na osnovi priporočilnih objektov.

```
1 function naOsnoviPriporočilnihObjektov (U, I)
  Input  : Množica uporabnikov U in priporočilnih objektov I1
  Output: Najvišje rangirani priporočilni objekti I2
2 foreach priporočilni objekt i za katerega uporabnik u še nima
  preference do
3     foreach priporočilni objekt j za katerega uporabnik u že ima
      preferenco do
4         Izračunaj podobnost s med priporočilnima objektoma i in j;
5         Izračunaj tekoče povprečje preferenc uporabnika u za
          priporočilni objekt j, uteženo s podobnostjo s;
6         Vrni priporočilne objekte I2, ki so rangirani po tem uteženem
          povprečju;
7     end
8 end
```

3.3.2 Modeli s skritimi spremenljivkami

Modeli s skritimi spremenljivkami so alternativa metodam najbližjih sosedov, ki smo jih opisali. Te metode uporabljajo skrite spremenljivke, ki v realnosti ne obstajajo, vendar nam tipično pomagajo opisati odnose med pojavi. V našem primeru želimo modelirati odnose med uporabniki in izdelki v obliki preferenčnih vrednosti oz. ocen, ki jih uporabniki izrazijo za priporočilne objekte. Eden izmed najuspešnejših načinov realizacije takšnih modelov je matrična faktorizacija [12].

Vsaka preferenčna vrednost oz. ocena je lahko opisana s poljubno veliko množico skritih spremenljivk, ki so *zelo specifični za izbrano problemsko domeno*. Tako se na primer spremenljivke, ki opisujejo količino akcije v filmu ali pa kompleksnost likov v filmu, močno razlikujejo od spremenljivk, ki opisujejo preferenčno oceno za neko nastanitev. Cilj je pridobiti te skrite spremenljivke iz preferenčnih vrednosti in jih nato shraniti v model ter kasneje uporabiti

za napoved neznanih preferenčnih vrednosti oz. generiranje priporočil [12].

Matrična faktorizacija preslikuje uporabnike in priporočilne objekte v skupen prostor skritih spremenljivk dimenzije f. Interakcije med uporabniki in priporočilnimi objekti lahko modeliramo kot notranje produkte vektorjev v tem prostoru. Vsak priporočilni objekt i je predstavljen z vektorjem $q_i \in \mathbb{R}^f$ in vsak uporabnik u je predstavljen z vektorjem $p_u \in \mathbb{R}^f$.

Za nek poljuben priporočilni objekt i vsak element vektorja q_i predstavlja vrednost, ki pove, do kakšne mere ta priporočilni objekt vsebuje neko skrito spremenljivko oz. skrito lastnost. Višja pozitivna vrednost pomeni večjo prisotnost te skrite lastnosti, medtem ko nižja ali negativna vrednost pomeni odsotnost te skrite lastnosti oz. spremenljivke v opisu priporočilnega objekta.

Za poljubnega uporabnika u elementi vektorja p_u predstavljajo interes uporabnika za priporočilne objekte, ki imajo visoko prisotnost ustreznih skritih spremenljivk, torej spremenljivk na enakem indeksu v q_i.

Skalarni produkt $q_i^T p_u$ prikazuje interakcijo med uporabnikom u in priporočilnim objektom i. Ta interakcija pomeni interes uporabnika za *skrite lastnosti* nekega priporočilnega objekta. Omenjeni interes se prikaže s približkom preferenčne vrednosti $\hat{r}_{ui} = q_i^T p_u$.

Največji izziv je izračun preslikave vsakega priporočilnega objekta in uporabnika v vektor skritih spremenljivk $q_i, p_u \in \mathbb{R}^f$. Ko imamo takšno preslikavo izračunano in shranjeno, lahko sistem preprosto oceni, kakšno preferenčno vrednost bo uporabnik pripisal nekemu poljubnemu priporočilnemu objektu.

Za reševanje omenjenega izziva se najpogosteje predlaga algoritem SVD (angl. *singular value decomposition*), ki je uveljavljena tehnika za pridobivanje skritih spremenljivk. Težava je v tem, da je pri skupinskem filtriranju potreben razcep matrike preferenčnih vrednosti, vendar vseh elementov matrike ne poznamo, kar pomeni, da ima mnogo manjkajočih vrednosti, ki jih ne moremo definirati kot 0. SVD tradicionalno ni definiran, ko vsi elementi matrike niso poznani. Zaradi tega SVD ni primeren in moramo uporabiti alternativen pristop, kot sta na primer stohastični gradientni spust (angl. *stochastic*

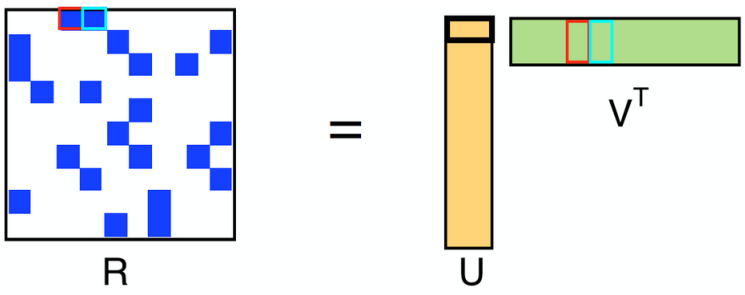
gradient descent, SGD) in ALS (angl. *alternating least squares*) [12].

Predlog za pristop, ki smo ga uporabili:

- Vsakega uporabnika in vsak priporočilni objekt opišemo z vektorjem skritih spremenljivk poljubne dolžine, ki pa mora biti vedno manjša od dimenzij osnovne matrike.
- Vsaka znana preferenčna vrednost je izračunana kot najboljši približek v skalarnem produktu med vektorjem skritih spremenljivk uporabnika, ki je to preferenčno vrednost izrazil, in vektorjem skritih spremenljivk priporočilnega objekta, za katerega je bila ta preferenčna vrednost izražena.
- Izračun neznanih preferenčnih vrednosti se izvede z enakim skalarnim produktom.
- Kvadratna napaka je mera izgube.

Aplikacija opisanega splošnega pristopa v našem priporočilnem sistemu je sledeča:

- Matriko preferenčnih vrednosti R razbijemo na produkt matrike uporabnikov U in matrike priporočilnih objektov V, kot prikazuje slika 3.2.



Slika 3.2: Razcep delno definirane matrike preferenčnih vrednosti R [26].

- Vsaka vrstica matrike V oz. stolpec matrike V^T predstavlja vektor skritih spremenljivk priporočilnega objekta q_i.

- Vsaka vrstica matrike U predstavlja vektor skritih spremenljivk uporabnika p_u.

- Matrika R je dimenzij $n \times m$, matrika U je dimenzij $n \times rang$, matrika V pa dimenzij $rang \times m$. *Rang* je poljubno izbrano celo število, ki določa dolžino vektorja oz. število skritih spremenljivk v vektorju. Tipično večje število skritih spremenljivk bolj natančno opiše relacijo, vendar po določeni meji razmerje med računsko zahtevnostjo in izboljšanjem rezultatov ni več ugodno. Parameter *rang* je za velike sisteme, kjer je matrika R velika, vedno zelo manjši od m in n ($rang << m, n$).

Matrika R v eni dimenziji predstavlja uporabnike, v drugi pa priporočilne objekte. Vsaka vrednost preseka vrstice in stolpca predstavlja preferenčno vrednost uporabnika vrstice i za priporočilni objekt stolpca j. Kot že omenjeno, je pomembno, da ta matrika ni samo delno prazna, ampak delno definirana, kar pomeni, da mankajočih vnosov ne smemo interpretirati kot 0, ampak kot neznanke. Omenili smo, da standardne metode razcepa v takšnih primerih niso primerne, zato ne moremo uporabiti metode SVD.

Razcep se mora izvesti samo z uporabo znanih preferenčnih vrednosti. To se rešuje tako, da sistem poišče vektorje uporabnikov in priporočilnih objektov s takšnimi latentnimi faktorji, da je kvadratna napaka skalarnih produktov, glede na znane preferenčne vrednosti, najmanjša. Minimizacija kvadratne napake na množici znanih preferenčnih vrednosti se izvede z enačbo [12]:

$$\min_{p,q} \sum_{u,i \in K} (r_{ui} - q_i^T p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2) \tag{3.1}$$

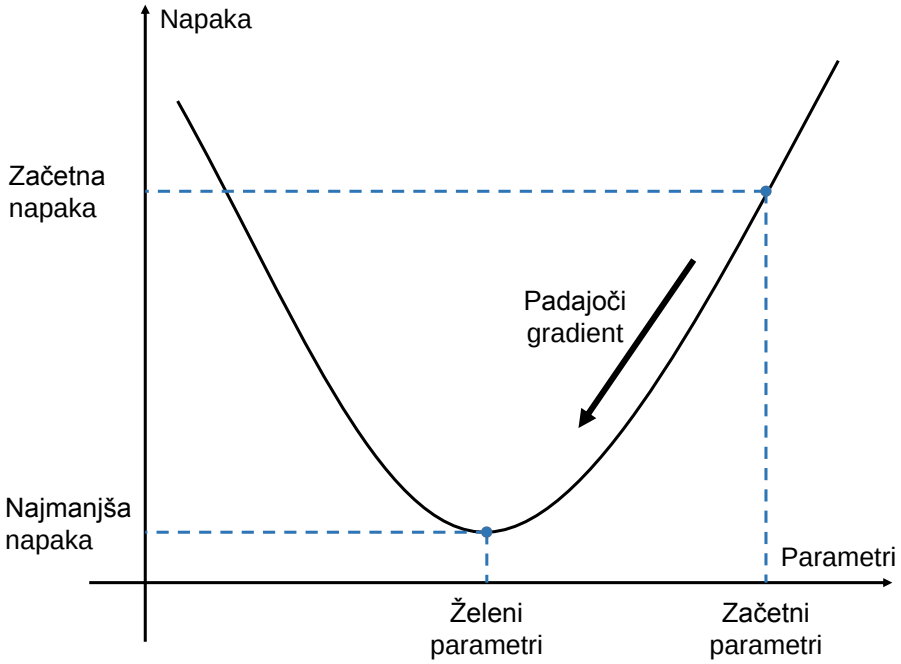
V enačbi (3.1) je K množica parov uporabnikov in priporočilnih objektov (u, i), za katere je preferenčna vrednost r_{ui} znana [12]. Sistem se nauči in zgradi model s tem, ko vektorje latentnih faktorjev ustvari tako, da produkte

teh vektorjev približamo že znanim preferenčnim ocenam. Končni cilj je posplošitev vektorjev z namenom napovedovanja neznanih preferenčnih vrednosti. Zaradi tega sistem ne sme prekomerno prilagajati vektorjev znanim preferenčnim vrednostim (angl. *overfitting*). To doseže z regularizacijo, ki jo uravnavamo z regularizacijsko konstanto λ v enačbi (3.1).

Minimizacija enačbe.

Minimizacijo enačbe (3.1) lahko izvedemo z uporabo algoritma stohastičnega gradientnega spusta (SGD) ali algoritma Alternating Least Squares (ALS).

Stohastični gradientni spust (SGD). Gradientni spust je standardna matematična optimizacijska metoda, ki deluje na osnovi računanja odvodov. Funkcija napake (3.1), ki jo želimo minimizirati, mora biti zato diferenciabilna. Pristop je prikazan na sliki 3.3.



Slika 3.3: Prikaz gradientnega spusta.

V opisanem primeru matrične faktorizacije algoritem iterira čez vse znane preferenčne vrednosti r_{ui}. Za vsako vrednost sistem izračuna približek r_{ui} in izračuna napako $e_{ui} = r_{ui} - q_i^T p_u$. Nato algoritem modificira skrite spremenljivke glede na faktor hitrosti učenja (angl. *learning rate*) v obratni smeri gradienta ter upošteva regularizacijo:

- $q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$,
- $p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$,

kjer γ predstavlja faktor hitrosti učenja, λ pa regularizacijski parameter. Ta algoritem zapišemo s psevdokodo:

Algoritem 3: Matrična faktorizacija z metodo stohastičnega gradientnega spusta.

```
1  function SGD (matrika ocen R):
2  Matrika uporabnikov U ← initSmallRandom();
3  Matrika priporočilnih objektov V ← initSmallRandom();
4  for iteracija in numIterations do
5      for uporabnik u in U.height do
6          for priporočilni objekt i in V.width do
7              napaka ← R[u,i] − U[u,:] · V[:,i];
               // prilagodi skrite spremenljivke U[u,:]  in
               V[:,i]
               // rang je število skritih spremenljivk v
               vektorju
8              for f in rang do
9                  U[u,f] = U[u,f] + γ · (napaka · I[f,i] − λ · U[u,f]);
10                 V[f,i] = V[f,i] + γ · (napaka · U[u,f] − λ · V[f,i]);
11             end
12         end
13     end
14 end
```

V enačbi (3.1) sta tako q_i kot p_u neznanki, zato funkcija ni konveksna. To rešujemo tako, da neznanke na začetku inicializiramo z naključnimi vrednostmi. SGD v osnovni različici popravlja po en vektor skritih spremenljivk naenkrat.

Alternating Least Squares (ALS). Algoritem ALS izmenjuje med fiksiranjem q_i-jev in p_u-jev. Na začetku inicializiramo eno izmed matrik z naključnimi vrednostmi. S tem pretvorimo enačbo na kvadratni optimizacijski problem, ki ga lahko sistem reši optimalno. Ko fiksiramo vse p_u-je (celotno matriko U), sistem ponovno izračuna vse q_i-je (celotno matriko V) z reševanjem z metodo najmanjših kvadratov in nato obratno. S tem se enačba (3.1) poenostavlja, dokler ne konvergira [12].

Algoritem ALS zapišemo s psevdokodo:

Algoritem 4: Matrična faktorizacija z metodo ALS.

```
1 function ALS (Matrika ocen R):
2 Matrika priporočilnih objektov V ← initSmallRandom();
3 Matrika uporabnikov U;
4 for iteracija in numIterations do
      // f je enačba (3.1)
5     Izračunaj matriko U, ki minimizira f, za fiksirano matriko V;
      // več vektorjev matrike prilagajaj vzporedno
6     Izračunaj matriko V, ki minimizira f, za fiksirano matriko U;
      // več vektorjev matrike prilagajaj vzporedno
7 end
```

Takšen algoritem je kompleksnejši kot stohastični gradientni spust, vendar je v sistemih, ki podpirajo vzporedno računanje, mnogo bolj ustrezen. Sistem lahko namreč izračuna vsak q_i neodvisno od ostalih faktorjev priporočilnega objekta in izračuna vsak p_u neodvisno od ostalih faktorjev uporabnika. To pomeni, da lahko teoretično v sistemu s 100 jedri izračunamo po 100 vektorjev q_i ali p_u naenkrat, in ne samo po enega. Zaradi tega se lahko algoritem **v**

zelo veliki meri izvaja vzporedno. To se sklada s paradigmo tehnologij velikih podatkovnih množic, ki smo jih opisali. Zato je ta algoritem primeren za uporabo v takšnih sistemih in smo ga tudi uporabili v našem strežniku za strojno učenje v oblaku, ki je razvit za uporabo z velikimi podatkovnimi množicami [12].

3.3.3 Priporočanje podobnih priporočilnih objektov na osnovi vsebinskega filtriranja

Poleg priporočil za uporabnika, kjer uporabimo metodo matrične faktorizacije, smo želeli implementirati tudi priporočanje podobnih nastanitev, kjer smo za osnovno idejo uporabili prej opisano vsebinsko filtriranje. Razlika pri naši implementaciji je v tem, da se v tej fazi nismo osredotočili na uporabnika in zato nismo povezali uporabniškega profila s profilom nastanitev, ampak smo ustvarili samo profile nastanitev in na podlagi lastnosti posamezne nastanitve priporočili univerzalno podobne. Takšen algoritem zahteva zbiranje podatkov, ki opisujejo nek priporočilni objekt oz. nastanitev z namenom kreacije vsebinskih profilov teh nastanitev. Uporabili smo pripadajoče filtre in geografske koordinate vsake nastanitve.

Te podatke smo pridobili v obliki datotek .csv, kot izvoz iz podatkovne baze razvijalca spletnega mesta `http://www.ecobnb.com`. Vsak filter je predstavljen s celoštevilsko vrednostjo, kjer določena vrednost pomeni unikaten filter (npr. *123* je unikatna celoštevilska predstavitev filtra *internet*). Geografski koordinati dolžine in širine pa sta predstavljeni z vrednostmi Double.

Filtri in izračun podobnosti. V prvi fazi smo vse filtre, ki pripadajo določeni nastanitvi, pretvorili v binarni vektor filtrov, ki opisuje dano nastanitev. Takšen vektor je vedno enake dolžine, in sicer tolikšne, kolikoršna je največja celoštevilska vrednost filtra v bazi. Vsaka enica v takšnem vektorju predstavlja prisotnost filtra na tem indeksu, vsaka ničla pa odsotnost filtra na tem indeksu. Indeks v vektorju predstavlja osnovno celoštevilsko vrednost

filtra. Tako na primer vektor $[0, 1, 1, 0]$ opiše nastanitev, ki ima filtra 2 in 3, nima pa filtrov 1 in 4.

V naslednjem koraku smo te binarne vektorje uporabili za izračun podobnosti med dvema nastanitvama. Uporabili smo Jaccardov koeficient za izračun podobnosti.

Zaradi binarnih vhodnih vektorjev, s katerimi predstavimo lastnosti nastanitve, je bolj kot uporaba kosinusne podobnosti smiselna uporaba Jaccardovega koeficienta podobnosti, ki je definiran z enačbo (3.2).

$$Jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{3.2}$$

V enačbi (3.2) A predstavlja množico filtrov prve nastanitve, B pa množico filtrov druge nastanitve. Presek predstavlja število filtrov, ki se pojavijo v obeh množicah. Rezultat $Jaccard(A, B)$ je definiran na intervalu $[0, 1]$, kjer višja vrednost pomeni večjo podobnost.

Geografske koordinate in evklidska razdalja. Poleg opisanih filtrov smo zbirali tudi podatke o geografskih koordinatih, ki smo jih smiselno uporabili za izračun podobnosti z uporabo evklidske razdalje. Podatki so omejeni na nastanitve na območju srednje in JV Evrope, torej na relativno ozkem geografskem območju. Koordinate so osnovane na storitvi Google Maps, ki implementira valjno kartografsko projekcijo Mercatorja [9]. Ker so koordinate v dvodimenzionalnem prostoru, smo uporabili klasično enačbo za evklidsko razdaljo (3.3).

$$distance = d(p, q) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2} \tag{3.3}$$

V enačbi (3.3) sta p in q točki v dvodimenzionalnem prostoru. Rezultat razdalje je skalar, ki je omejen z intervalom $[0, \infty)$, kjer 0 pomeni popolnoma enaki koordinati, vsaka večja vrednost pa bolj oddaljeni nastanitvi.

Združena podobnost. V zadnji fazi smo za izračun podobnosti želeli združiti obe izračunani vrednosti, torej evklidsko razdaljo in Jaccardovo

podobnost, v enotno oceno podobnosti med dvema nastanitvama. Zaradi različnih intervalov omejenosti posameznih ocen je potrebno najprej izvesti normalizacijo. Poleg tega je pomembna razlika v tem, da pri Jaccardovi podobnosti večja vrednost pomeni višjo podobnost, med tem ko pri evklidski razdalji manjša vrednost pomeni višjo podobnost, kar se tiče lokacije.

Izziv, ki se pri tem problemu pojavi in smo ga že nakazali, je ustrezno uravnovešenje obeh individualnih ocen podobnosti. Zagotoviti je potrebno, da nobena izmed njiju ne prevlada.

Kot omenjeno je potrebno upoštevati, da pri evklidski razdalji nižja vrednost pomeni višjo podobnost, kar je ravno obratno kot pri Jaccardovem koeficientu. Zaradi tega moramo razdaljo spremeniti v podobnost, kar dosežemo z enačbo (3.4).

$$similarity_E = \frac{1}{1 + d(p, q)} \tag{3.4}$$

V enačbi (3.4) dobimo v primeru najnižje možne evklidske razdalje 0 (enake koordinate) največjo podobnost 1. V nasprotnem primeru z velikimi vrednostmi evklidske razdalje podobnost relativno hitro konvergira proti 0.

V zadnjem koraku obe podobnosti združimo v združeno oceno podobnosti z enačbo (3.5).

$$similarity = \frac{(w_c \cdot sim_c) + (w_e \cdot sim_e)}{w_c + w_e} \tag{3.5}$$

V enačbi (3.5) sta w_c in w_e uteži, s katerima uravnavamo prispevek posamezne podobnosti k skupni podobnosti.

Poglavje 4

Uporaba za priporočanje v nastanitveni dejavnosti

Nastanitvena dejavnost je poleg spletnih trgovin najbolj izpostavljena panoga spletnega trženja in posledično zahteva velika vlaganja za pridobitev konkurenčne prednosti. Že v uvodu smo nakazali, da so priporočilni sistemi in dinamično uravnavanje cen ključnega pomena, saj lahko pomenijo znatno izboljšanje poslovnih rezultatov.

4.1 Opis produkcijskega problema

Spletno mesto `http://www.ecobnb.com` se specializira v ponudbo ekoloških nastanitev, ki v zadnjem času postajajo vse bolj priljubljena tržna niša. Ekološke nastanitve morajo dosegati vsaj pet od desetih pogojev, ki jih definira ponudnik Ecobnb. Primer takšnih pogojev so: uporaba sončne energije, filtriranje in ponovna uporaba vode itd.

Soočimo se s problemom razvoja in implementacije priporočilnega sistema, ki ga želimo implementirati v realnem produkcijskem okolju v njihovi spletni aplikaciji. Sistem ponudimo kot storitev v oblaku, ki jo lahko spletna aplikacija uporablja kot storitev REST. Podatke zbiramo v realnem času in s tem periodično povečujemo učno množico. Sistem mora ponujati možnost

porazdeljenega delovanja in skalabilnosti ter mora arhitekturno ustrezati potencialni uporabi z velikimi podatkovnimi množicami na gruči računalnikov.

Sistem ocenimo z uporabo statistične mere Precision@k in v zadnji fazi predlagamo implementacijo prikaza priporočil nastanitev v realnem času na spletnem mestu `http://www.ecobnb.com`.

4.2 Opis podatkov in transformacij

Kot opisano v poglavju 3, na spletnem mestu v realnem času beležimo vse akcije uporabnikov. Ključno pri tem je, da identificiramo unikatne uporabnike in beležimo akcije, ki pripadajo istemu uporabniku, prek daljšega časovnega obdobja. To razvijalci spletne aplikacije dosežejo z uporabo dolgo trajajočih piškotkov.

Podatke želimo pridobivati v obliki, ki bi že v osnovi ustrezala tipični "*uporabnik – akcija – priporočilni objekt*" obliki, ki je najpogosteje v uporabi v algoritmih skupinskega filtriranja. S tem namenom razvijemo in uporabimo storitev REST, ki je opisana v poglavju 3. Storitev v skladu z dogodkovno paradigmo sprejema podatke v realnem času. Vsakič, ko se na spletni strani zgodi akcija uporabnika, se storitvi pošlje zahtevek HTTP POST s sledečo obliko:

```
'properties' : {
        'action' : <String>,
        'resource': <String>,
        'parameter': <String/Int/Double/List/nested JSON>,
        'tracking_id': <String>,
        'timestamp': <date_time_in_ISO_format>
}
```

V zgoraj navedenem zahtevku JSON imamo naslednje ključne parametre:

- action: akcija, ki jo nek uporabnik izvede na spletni strani. Vedno jo navedemo v obliki *site-action*, kjer *site* predstavlja eno izmed strani

oziroma podstrani spletnega mesta ecobnb.com, *action* pa predstavlja dejansko akcijo uporabnika,

- resource: objekt, nad katerim je akcija izvršena. Tipično gre za nek objekt na spletni strani, ki je ustrezno predstavljen tudi v podatkovni bazi,

- parameter: dejanska vrednost *resource-a*, nad katerim se izvrši akcija. Definirani so pričakovani tipi vrednosti,

- tracking_id: unikaten identifikator uporabnika, ki je izvedel akcijo. Ker uporabniki večinoma niso prijavljeni v spletno stran, gre za dolgo trajajoč id piškotka,

- timestamp: čas, ob katerem se je dogodek izvršil na strani odjemalca, torej na spletnem mestu ecobnb.com.

Vnaprej smo definirali množico potencialno izvedljivih akcij na odjemalcu. Poleg tega definiramo tudi množico objektov, nad katerimi se te akcije lahko izvedejo. Na ta način imamo pričakovano in vnaprej definirano obliko podatkov, kjer lahko zlahka zaznamo anomalije.

V drugi fazi smo te podatke prilagodili za uporabo z algoritmom ALS, opisanim v poglavju 3. Algoritem iz podatkov najprej zgradi matriko, kjer ena os predstavlja uporabnike, druga pa nastanitve. Vsaka vrednost v matriki predstavlja preferenčno vrednost uporabnika te vrstice za nastanitev tega stolpca. V realnih sistemih se seveda dogaja, da isti uporabniki izkažejo večje število različnih preferenc za neko nastanitev. Tipično si jo najprej ogledujejo, nato odprejo poizvedbo, jo potencialno prekličejo in morda kasneje pošljejo. Zaradi tega v tej matriki ne moremo predstavljati le ene preferenčne vrednosti, ampak je ta problem v MLlib implementaciji rešen tako, da je vrednost v matriki vedno najvišja med vsemi izraženimi preferenčnimi vrednostmi nekega unikatnega uporabnika nad to nastanitvijo (angl. *maximum*).

Primer: *Uporabnik X* si najprej trikrat ogleda *nastanitev Y*, nato odpre okno za poizvedbo, ga zapre, si nastanitev še dvakrat ogleda, ponovno odpre okno za poizvedbo in poizvedbo pošlje.

V takem primeru preferenčna vrednost *uporabnika X* za *nastanitev Y* v matriki predstavlja vrednost 5.0, saj ohranimo najvišjo izmed izkazanih preferenc.

V našem primeru je šlo za implicitne akcije, saj spletno mesto ne podpira eksplicitnih izražanj preferenc, kot so na primer ocene. V tej fazi smo torej implicitne akcije pretvorili v virtualne ocene oziroma preferenčne vrednosti s sledečim preslikovanjem:

- accommodation-view ⇒ vrednost 2.0
  Ogled nastanitve predstavlja osnoven izkaz preference uporabnika in ga ustrezno preslikamo v nizko vrednost.

- accommodation-inquire_open ⇒ vrednost 4.0
  Če uporabnik po ogledu nastanitve odpre okno za kontakt s ponudnikom, to predstavlja večjo preferenco in to akcijo preslikamo v srednje visoko vrednost.

- accommodation-inquire_send ⇒ vrednost 5.0
  V kolikor uporabnik pošlje poizvedbo za nastanitev, to kaže na najvišji interes in preferenco, zato takšno akcijo preslikamo v najvišjo vrednost.

- accommodation-inquire_cancel ⇒ vrednost 4.0
  Če uporabnik zapre okno za poizvedbo, to tipično pomeni, da ima visok interes, vendar se nato v zadnjem koraku premisli. Takšen primer preslikamo v srednje visoko vrednost.

4.3 Poskusi in rezultati

Priporočilni sistemi so tradicionalno težavni za objektivno ocenjevanje [8], saj uporabniki zelo subjektivno reagirajo na priporočila. Zaradi manjše razvojne

skupine in omejitev pri razvoju vzporedna vpeljava priporočilnega sistema in beleženje uporabe v realnem času nista bila možna. Posledično smo se odločili za statistično mero *Precision@k*, ki smo jo uporabili s prečnim preverjanjem (angl. *cross-validation*) na zbranih podatkih in nam služi kot oporna mera za napredovanje ali nazadovanje sistema med verzijami.

Precision@k. Gre za eno izmed standardnih mer v sistemih za pridobivanje informacij (kot so priporočilni sistemi). Preciznost je definirana kot delež relevantnih pridobljenih priporočilnih objektov med vsemi pridobljenimi priporočilnimi objekti, kar prikazuje enačba (4.1) [17]:

$$Preciznost = \frac{\#(\text{relevantni pridobljeni priporočilni objekti})}{\#(\text{vsi pridobljeni priporočilni objekti})} \tag{4.1}$$

V sistemih, kot so iskalniki ali priporočilni sistemi, je pomembno, da se boljši rezultati pojavijo prej v skupnem rezultatu. To je tipično vidno pri iskalnikih kot so Google, kjer uporabniki večinoma klikajo le zadetke na prvi strani rezultatov. Podobno velja za priporočilni sistem, kjer imamo v realni aplikaciji omejitev števila prikazanih priporočil. Zelo pomembno je torej, da se najboljša (torej relevantna) priporočila pojavijo med omejenim številom priporočenih. Posledično želimo merjenje preciznosti izvesti na omejenem številu vrnjenih priporočil, kar imenujemo *Precision@k*, kjer k pomeni število prvih k vrnjenih priporočil, za katere merimo preciznost (npr. Precision@10 pomeni merjenje preciznosti med prvimi desetimi vrnjenimi rezultati). Prednost te mere je v tem, da ne potrebujemo znanja o številu vseh relevantnih priporočilnih objektov. Na drugi strani pa je ključna slabost večja nestabilnost, saj ima število vseh obstoječih relevantnih priporočilnih objektov velik vpliv na rezultat [17]. Opisano definicijo prikazuje enačba (4.2):

$$Preciznost@k = \frac{\#(\text{relevantni priporočeni objekti med k})}{\text{k priporočenih priporočilnih objektov}} \tag{4.2}$$

Priporočilni sistem smo ocenili z uporabo opisane mere Precision@k tako, da smo izvedli prečno preverjanje. Množico vseh podatkov smo naključno razdelili v učno in testno množico, kjer učna množica predstavlja 80% vseh

podatkov, testna pa 20%. To smo izvedli 5-krat, kjer smo vsakič vzeli drugačno množico 80% primerov kot učne in preostalo množico 20% primerov kot testne podatke. Nato smo za vsakega uporabnika merili, koliko izmed k vrnjenih priporočil, ki jih vračamo glede na model, ustvarjen z učno množico, se pojavi tudi v testni množici [8].

Razred Evaluator. V Prediction.IO strežniku je za ocenjevanje potrebno sprogramirati poseben razred imenovan *Evaluation*, ki je ločen od pogona in ni nujno potreben za delovanje. Kljub temu je del standardne arhitekture DASE (E = Evaluation).

Parametri algoritma. Zaradi matematične narave algoritma ALS, opisanega v poglavju 3, ki uporablja skrite spremenljivke, razvijalec ne more vedeti, koliko skritih spremenljivk bo najbolje opisalo matriko s preferenčnimi vrednostmi. Dolžino vektorjev skritih spremenljivk določimo s parametrom *rang*, ki je tipično vedno zelo manjši od dimenzij osnovne matrike preferenčnih vrednosti ($rang << m, n$), sploh ko je ta matrika zelo velika. Smiselno je testirati tudi različno število iteracij za izgradnjo učnega modela, torej iteracij, v katerih se skrite spremenljivke spreminjajo in prilagajajo znanim preferenčnim vrednostim. To lahko določamo s parametrom *numIterations*. Posploševanje modela in izogibanje prevelikemu prilagajanju učnim podatkom se uravnava z regularizacijskim parametrom λ, ki je tipično majhna vrednost, saj s tem omejimo velikost skritih spremenljivk.

Parametri ocenjevanja. Poleg parametrov algoritma lahko uravnavamo tudi parametre ocenjevanja. Parameter *kFold* predstavlja število, ki ga uporabimo za delitev podatkov na učno in testno množico, ter obenem število iteracij za to delitev. V našem primeru smo parameter *kFold* nastavili na $kFold = 5$, kar pomeni, da gre vedno za fiksno 5-kratno delitev podatkov v razmerju 80% učni, 20% testni. Parameter *queryNum* pa pove, koliko poizvedb generiramo za vsakega uporabnika. Pri metriki Precision@k je v skladu s tem potreben pogoj $k <= queryNum$. Tretji parameter

je *threshold*, ki predstavlja mejo relevantnosti za našo mero Precision@k. S tem parametrom uravnavamo, ali so za sistem relevantni samo nakupi nastanitev ($threshold = 4.0$), ali pa so relevantni tudi ogledi nastanitev ($threshold = 2.0$). Mejo relevantnosti je smiselno nastaviti tako, da ni prenizka, saj bi s tem povzročili relevantnost prevelikega števila dogodkov. Obenem tudi ne sme biti previsoka, saj s tem lahko povzročimo anomalijo, kjer noben dogodek ni relevanten.

Sistem testiramo z uporabo omenjenega razreda Evaluator, kjer smo ocenjevanje izvedli z različnimi vrednostmi opisanih parametrov algoritma. Namen tega je bil poiskati množico parametrov algoritma, pri katerih se sistem najbolje obnese.

Za parametre ocenjevanja smo izbrali smiselne vrednosti glede na problem, s katerim se soočamo. Za prag (angl. *threshold*) smo izbrali vrednost 2.0, kar pomeni, da kot relevantni dogodek upoštevamo ogled in rezervacijo nastanitve. S tem smo določili večino akcij kot relevantne, kar je smiselno glede na majhno povprečno število akcij na uporabnika. V kolikor sistem priporoči nastanitev, nad katero je bila dejansko izvedena kakršnakoli akcija (ogled, rezervacija, nakup itd.), to pomeni uspešno priporočilo. Parameter k smo nastavili na vrednosti 3, 5 in 10, kar ustreza omejitvi na spletni strani, kjer so 3 nastanitve vidne takoj, ko uporabnik izvede iskanje. V primeru premikanja po prvi strani zadetkov pa lahko uporabnik vidi do 10 nastanitev.

Testiranje. Izkazalo se je, da so rezultati najboljši z naslednjimi parametri algoritma:

- $rang = 20$,
- $numIterations = 5$,
- $\lambda = 0.01$.

Ob teh parametrih smo izbrali naslednje smiselne vrednosti parametrov ocenjevanja:

- $kFold = 5$,
- $k = 3, 5, 10$, kjer je k skupno število vrnjenih priporočil, za katere merimo preciznost ($Precision@k$),
- $threshold = 2.0$.

Najboljše vrednosti za parametre *rang*, *numIterations* in λ so določene eksperimentalno. Število iteracij prilagajanja faktorjev in dolžina vektorjev latentnih faktorjev oba bistveno vplivata na čas izvajanja učenja in evalvacije sistema. Nad določeno mejo se čas izvajanja bistveno povečuje, medtem ko se Precision@k skorajda ne spreminja ali pa celo rahlo poslabša. Izbrali smo torej vrednosti, pri katerih je čas izvajanja sprejemljiv za produkcijski sistem, obenem pa se rezultati z večjima vrednostima parametrov ne izboljšujejo.

Najboljši rezultat *Precision@k*, ki smo ga dosegli z uporabo optimalnih parametrov algoritma, je **0,1879**. To pomeni, da je v 19% primerov priporočenih nastanitev uporabnik tudi dejansko imel interakcijo s priporočeno nastanitvijo. Interakcija pomeni izvedbo ene izmed možnih akcij (ogled, rezervacija, nakup itd.). Rezultati ocenjevanja so prikazani v tabeli 4.1.

rang	numIterations	Precision@3	Precision@5	Precision@10
5	5	1,74%	1,60%	1,25%
5	10	5,38%	4,09%	2,68%
10	5	8,90%	6,43%	4,05%
10	10	17,66%	11,48%	6,29%
20	5	**18,79%**	12,18%	6,69%
20	10	16,69%	11,01%	6,16%
Naključno priporočanje		0,97%	0,94%	0,92%

Tabela 4.1: Evalvacija s parametri ocenjevanja $threshold = 2.0$, $kFold = 5$ in $k = 3, 5, 10$.

Interpretacija rezultatov. Kot smo že omenili, so priporočilni sistemi težavni za objektivno ocenjevanje, saj uporabniki v realnem svetu zelo subjektivno reagirajo na priporočila. To se tipično kaže v tem, da uporabniki klikajo na priporočila že samo zato, ker so na voljo, četudi so ta naključna.

V opisanem primeru problema smo za prag relevantnosti izbrali akcijo *ogled* ($threshold = 2.0$) in sicer zato, ker ima povprečen unikaten uporabnik le 2-3 akcije v zgodovini. V kolikor bi za prag relevantnosti izbrali akcijo *rezervacija* ($threshold = 4.0$), bi bilo zelo smiselno rezultate postaviti v kontekst z normalizacijo. V tem primeru bi bila namreč povprečna vrednost v števcu enačbe 4.2 manj kot 1, vendar lahko za namen interpretacije predpostavimo, da je v števcu kar vrednost 1, saj uporabnik ponavadi po večih ogledih rezervira le eno nastanitev. Ob tej predpostavki mera *Precision@k* postane pesimistična, saj nikoli ne more doseči vrednosti 1 (100% natančnost). Zato rezultate normaliziramo, tako da dobljene natančnosti delimo z največjo možno vrednostjo *Precision@k* za izbrani k. Najboljši rezultati bi ob tej interpretaciji postali:

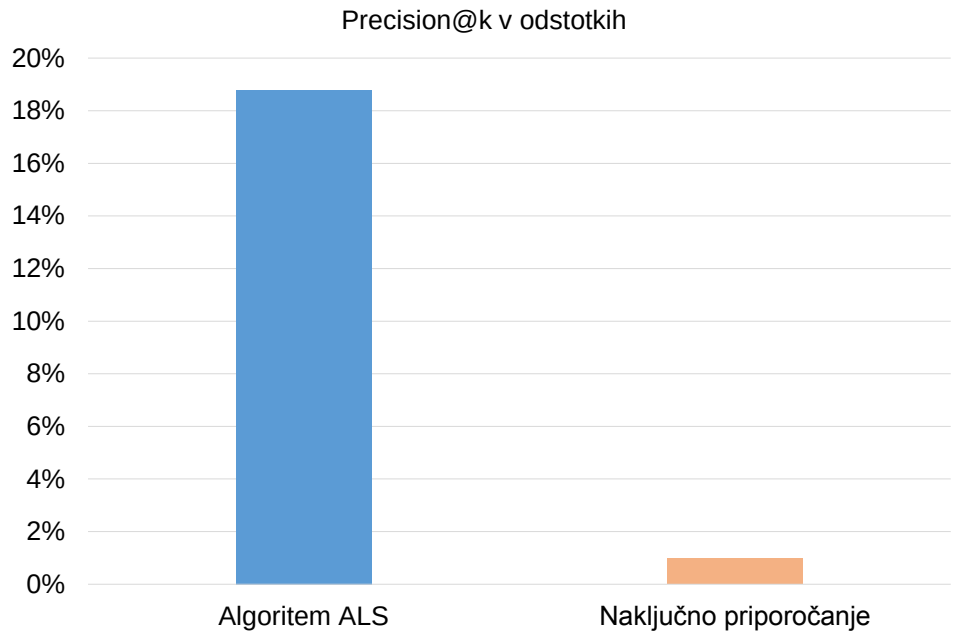
$$\begin{aligned} Precision@3 &= \frac{0,1879}{\frac{1}{3}} = 0,5637 \rightarrow 56,37\% \\ Precision@5 &= \frac{0,1218}{\frac{1}{5}} = 0,6090 \rightarrow 60,90\% \\ Precision@10 &= \frac{0,0669}{\frac{1}{10}} = 0,6690 \rightarrow 66,90\% \end{aligned} \tag{4.3}$$

V tabeli 4.1 opazimo zmanjševanje natančnosti z večanjem parametra k, torej števila priporočenih nastanitev. Glede na majhno povprečno število akcij na unikatnega uporabnika (med 2 in 3) je to smiselno, saj ponavadi relevantne nastanitve pravilno napovemo že zgodaj, torej med prvimi tremi ali petimi priporočenimi ($k = 3, 5$). Ker v povprečju s prvima dvema pravilno napovedanima nastanitvama pokrijemo vse akcije povprečnega unikatnega uporabnika, lahko z večanjem števila priporočenih nastanitev vračamo samo še nerelevantne nastanitve. Posledično se natančnost z večanjem parametra k zmanjšuje.

Rezultati natančnosti napovedi v območju 10-20% sami po sebi ne izgledajo najboljši, vendar se je potrebno zavedati, da v produkcijskem okolju potencialno 10% povečanje prodaje pomeni velik poslovni uspeh. Amazon je po uvedbi svojega priporočilnega sistema, ki velja za enega najboljših, dosegel 29% povečanje prodaje v primerjavi z letom prej. Del te izboljšave lahko pripišemo trendu povečevanja spletnih nakupov, del pa uvedbi priporočilnega sistema [16].

Primerjava z naključnim priporočanjem. Naš najboljši rezultat je smiselno primerjati z naključnim priporočanjem, kjer uporabimo enake parametre ocenjevanja. Na ta način jasno prikažemo izboljšanje kvalitete priporočil. S tem namenom smo v istem strežniku Prediction.IO razvili preprost sistem naključnega priporočanja, ki le izbere naključne nastanitve iz baze in jih vrne kot priporočilo. Takšen sistem se zaradi velikega števila priporočilnih objektov in uporabnikov pričakovano obnese zelo slabo in bistveno slabše od našega razvitega priporočilnega sistema.

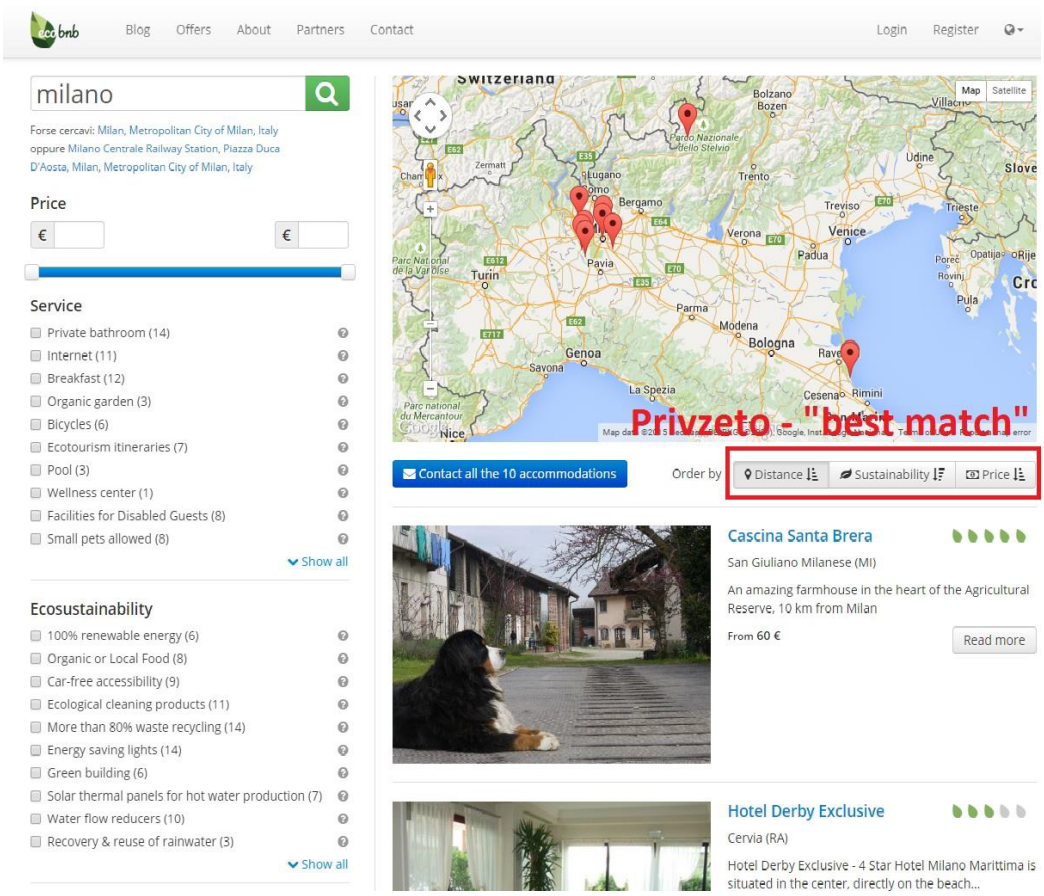
Primerjava je jasno razvidna na sliki 4.1, ki prikazuje oba rezultata z uporabo mere preciznosti (*Precision@k*), ki smo jo izbrali in opisali. Razberemo lahko, da je naš sistem bistveno uspešnejši kot naključno priporočanje, saj dosegamo rezultat 19%, ki je bistveno višji od rezultata naključnega priporočanja, ki je pod 1%.



Slika 4.1: Primerjava rezultatov našega sistema in naključnega priporočanja z uporabo mere Precision@k. Rezultati so v odstotkih.

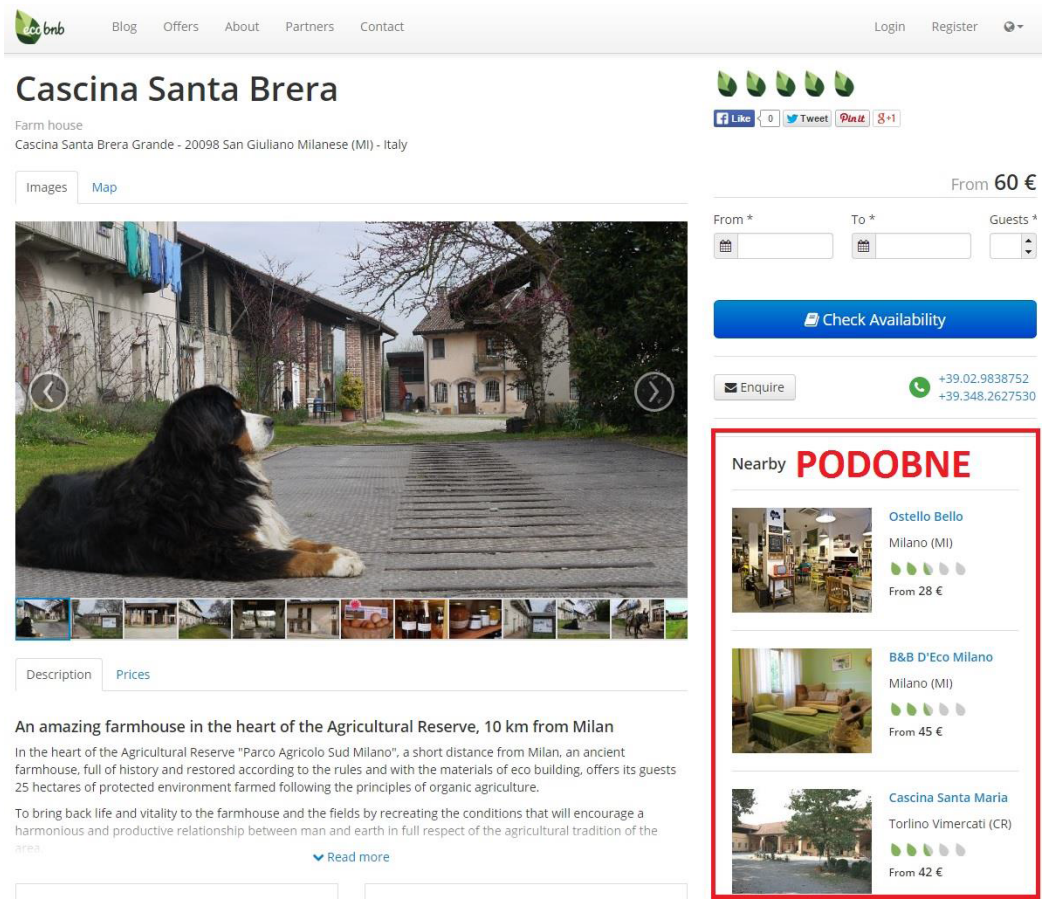
4.4 Implementacija na spletni strani

Opisan algoritem ALS je na spletni strani smiselno uporabiti za priporočanje nastanitev uporabniku, za katerega poznamo zgodovino. V primeru, da se na strani pojavi nov uporabnik, mu priporočimo najbolj priljubljene nastanitve. V kolikor pa je uporabnik že imel interakcijo s spletno stranjo, predlagamo implementacijo zavihka *"Order by: best match"*, ki bi bil privzeto izbran. Na ta način bi uporabnik ob iskanju privzeto dobil prikazan personaliziran seznam nastanitev. Takšen predlog je prikazan na sliki 4.2.



Slika 4.2: Prikaz koncepta predlagane implementacije priporočil za uporabnika na spletni strani.

Podobno bi drugi implementirani algoritem za priporočanje podobnih nastanitev implementirali na strani s podrobnostmi individualne nastanitve. Trenutno je na takšnem mestu implementiran prikaz najbližjih nastanitev, kot je prikazano na sliki 4.3. Na enakem mestu bi torej prikazovali najbolj podobne glede na izračunano podobnost, opisano v poglavju 3.



Slika 4.3: Prikaz koncepta predlagane implementacije priporočil podobnih nastanitev na spletni strani.

Poglavje 5

Zaključek

Soočili smo se z realnim problemom razvoja priporočilnega sistema v produkcijskem okolju.

V prvem poglavju smo opisali problem, ki zajema izbor platforme za strojno učenje v oblaku, zbiranje podatkov v produkcijskem sistemu in razvoj priporočilnega sistema, ki ga lahko ponudimo kot storitev. Zastavili smo cilj, ki je delujoč priporočilni sistem, ki omogoča porazdeljeno uporabo in zagotavlja skalabilnost za uporabo z velikimi podatkovnimi množicami.

Tekom drugega poglavja smo se seznanili s področjem velikih podatkov ter izpostavili ključne razlike v primerjavi s tradicionalno podatkovno analitiko. Analizirali smo najpomembnejše tehnologije na področju velikih podatkov (Apache Hadoop, HBase, Spark, MLlib, Elasticsearch), ki skupaj lahko sestavljajo zaokrožen sklad za strojno učenje. Spoznali smo nekaj izmed glavnih ponudnikov strojnega učenja v oblaku, jih primerjali in na podlagi primerjave izbrali najustreznejšega za naš problem.

V tretjem poglavju smo podrobneje razdelali področje priporočilnih sistemov, predstavili arhitekturo izbranega strežnika za strojno učenje v oblaku in obenem izpostavili paradigmo MVC v tej arhitekturi. Nato smo opisali storitev REST za zbiranje podatkov in natančno razdelali metodologijo. Opisali smo skupinsko filtriranje, algoritem matrične faktorizacije in optimizacijske metode. Nato smo predstavili lastno mero za podobnost na osnovi Jaccar-

dove podobnosti in evklidske razdalje.

Zadnje poglavje je izpostavilo podroben opis podatkov ter predvsem ocenjevanje našega sistema z uporabo statistične mere *Precision@k*, kjer smo dosegli najboljši rezultat 19%. Končali smo s predlogom možne implementacije v produkcijskem okolju.

5.1 Pomen

V diplomski nalogi smo rešili problem uporabe knjižnic za strojno učenje, ki že implementirajo osnovne verzije mnogih algoritmov, vendar se ne ukvarjajo z zbiranjem in hranjenjem podatkov ter komunikacijo z odjemalci. Z uporabo strežnika za strojno učenje Prediction.IO razvijemo produkcijsko storitev, ki zbira podatke, zbrane podatke uporabi v algoritmu matrične faktorizacije za izgradnjo napovednega modela ter komunicira z uporabniki. To storitev bo na preprost način uporabljal ponudnik ekoloških nastanitev Ecobnb.

Delo je v zadnji fazi razvoja, kjer se bo na spletni strani implementiral prikaz priporočenih nastanitev. Glede na rezultate evalvacije, kjer smo dosegli 19% Precision@k na že zbranih podatkih, pričakujemo izboljšanje poslovnih rezultatov podjetja Ecobnb. Rezultat je bistveno boljši kot pri naključnem priporočanju, ki doseže 1%. Končni uporabniki sistema bodo ljudje, ki bodo prek ponudnika Ecobnb iskali ekološke nastanitve na območju srednje in JV Evrope. Sistem bo tem končnim uporabnikom zmanjšal pregledovanje velikega števila nastanitev in poenostavil iskanje s smiselnimi priporočili. Obenem pričakujemo povečanje števila rezervacij nastanitev ter zvišanje zadovoljstva končnih uporabnikov, kar bi pomenilo poslovni uspeh podjetja Ecobnb.

V relativno kratkem času in na preprost način smo uporabili kompleksne algoritme strojnega učenja za priporočanje uporabniku na osnovi matrične faktorizacije. Poleg tega smo v obstoječ sistem dodali lasten algoritem za priporočanje podobnih priporočilnih objektov, ki je osnovan na Jaccardovi podobnosti (podobno kot Amazon-ov priporočilni sistem) in nadgrajen z upo-

rabo evklidske razdalje. Obe razdalji smo združili v eno podobnost. S tem smo pokazali, da implementacija priporočilnega sistema danes ne zahteva več toliko vlaganj in ekspertnega znanja kot v preteklosti.

5.2 Nadaljnje delo

Delo je še v aktivni fazi razvoja in ponuja kar nekaj možnosti nadaljnega dela. V prihodnosti so smiselne sledeče nadgradnje:

- uspešnost priporočanja zaradi unikatnosti podatkov in problema težko primerjamo z drugimi rešitvami. Znotraj lastnega sistema pa je smiselna implementacija več različnih algoritmov in njihova medsebojna primerjava na istih podatkih,
- uporaba matrične faktorizacije in algoritma ALS, prilagojenih za implicitne podatke,
- implementacija prikaza priporočil na spletni strani (v aktivnem razvoju),
- merjenje izboljšanja poslovnih rezultatov po implementaciji prikazovanja priporočil,
- zbiranje podatkov o akcijah, ki niso povezane z nastanitvami (npr. demografski profili),
- nadgradnja podatkov o nastanitvah s cenovnimi razredi, ki so tipično zelo pomembni.

Literatura

[1] Recommender systems, part 1: Introduction to approaches and algorithms. "http://www.ibm.com/developerworks/library/os-recommender1/os-recommender1-pdf.pdf". (Obiskano dne 17. 12. 2014).

[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, junij 2005.

[3] R. M. Bell, Y. Koren, and C. Volinsky. The bellkor solution to the netflix prize. http://www.netflixprize.com/assets/ProgressPrize2008_BellKor.pdf, 2009.

[4] N. Dimiduk, A. Khurana, and M.H. Ryan. *HBase in Action*. Manning, 2012.

[5] L. George. *HBase: The Definitive Guide*. O'Reilly Media, 2011.

[6] C. Gormley and Z. Tong. *Elasticsearch: The Definitive Guide*. O'Reilly Media, 2015.

[7] M. Hamstra, H. Karau, M. Zaharia, A. Konwinski, and P. Wendell. *Learning Spark: Lightning-Fast Big Data Analytics*. O'Reilly Media, Incorporated, 2015.

[8] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, januar 2004.

[9] R. Israel. Mercator's projection. `http://www.math.ubc.ca/~israel/m103/mercator/mercator.html`, januar 2003. (Obiskano dne 21. 8. 2015).

[10] D. Jannach and G. Friedrich. Tutorial: Recommender systems. `"http://ijcai13.org/files/tutorial_slides/td3.pdf"`, 2013. (Obiskano dne 17. 12. 2014).

[11] Y. Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 426–434, New York, NY, USA, 2008.

[12] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, avgust 2009.

[13] C. Lam. *Hadoop in Action*. Manning Publications, december 2010.

[14] J. Leskovec. Recommender systems: Content-based systems & collaborative filtering. `"http://web.stanford.edu/class/cs246/slides/07-recsys1.pdf"`, 2014. (Obiskano dne 17. 12. 2014).

[15] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, januar 2003.

[16] J. P. Mangalindan. Amazon's recommendation secret - fortune. `http://fortune.com/2012/07/30/amazons-recommendation-secret/`, julij 2012. (Obiskano dne 1. 9. 2015).

[17] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[18] P. Melville and V. Sindhwani. Recommender systems. In *Encyclopedia of machine learning*, pages 829–838. Springer, 2010.

[19] A. Rajaraman, J. D. Ullman, and J. Leskovec. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011.

[20] P. Resnick and H. R. Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[21] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM.

[22] T. Segaran. *Programming Collective Intelligence*. O'Reilly, first edition, 2007.

[23] G. Slapničar and B. Kaluža. Cloud-based recommendation system for e-commerce. `http://is.ijs.si/zborniki/2014_IS_CP_Volume-A_%28IS%29.pdf`, oktober 2014. (Obiskano dne 15. 8. 2015).

[24] G. Takács, I. Pilászy, B. Németh, and D. Tikk. Scalable collaborative filtering approaches for large recommender systems. *J. Mach. Learn. Res.*, 10:623–656, junij 2009.

[25] G. Takács and D. Tikk. Alternating least squares for personalized ranking. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 83–90, New York, NY, USA, 2012.

[26] B. Yavuz, X. Meng, and R. Xin. Scalable collaborative filtering with spark mllib — databricks. `https://databricks.com/blog/2014/07/`

23/scalable-collaborative-filtering-with-spark-mllib.html, julij 2014. (Obiskano dne 16. 8. 2015).

[27] H.-F. Yu, C.-J. Hsieh, I. Dhillon, et al. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 765–774. IEEE, 2012.

[28] D. Zachariah, M. Sundin, M. Jansson, and S. Chatterjee. Alternating least-squares for low-rank matrix reconstruction. *Signal Processing Letters, IEEE*, 19(4):231–234, april 2012.

[29] Y. Zhou, D. Wilkinson, R. Schreiber, and R. Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*, AAIM '08, pages 337–348, Berlin, Heidelberg, 2008.

[30] P. Zikopoulos, D. Deroos, C. Bienko, R. Buglio, and M. Andrews. *Big Data Beyond the Hype: A Guide to Conversations for Todays Data Center*. McGraw-Hill Education, 2014.

[31] P. Zikopoulos, C. Eaton, et al. *Understanding big data: Analytics for enterprise class hadoop and streaming data*. McGraw-Hill Osborne Media, 2011.